Submersible Remote Operated Vehicle (ROV) Journal

Members: Kow Ming Yuan, Arshul Garg, Soh Kai Xuan, Rajan, Mohit, David
Started on: 17 November 2022
Completed on: 19 January 2022

Members: Karthik, Jason, Sarah, Santhosh, Cheng Ju
Started on:

For full code and other resources, refer to the Github for this project:
https://github.com/Kmyming/vjcunderwaterbot

For project videos and photos, refer to the google drive:
https://drive.google.com/drive/u/1/folders/1dox0n6uJKZHGL2Pta8rMBiVloc-5PMgp

General Format of each section:

1. **Summary of how component works and aim**
   (e.g. controller: controlling vehicle motion through WiFi control to wirelessly maneuver the vehicle. 3 different controls, ___, ___, ___. For joystick control, calibration was done through ____.)
2. **List our all materials required/electrical components needed**
3. **(For coding): Write out general logic in pseudo code -- point form or flow chart**
   (e.g. WiFi: Server sends HTTP GET request to client -> Client receives HTTP GET request -> Client splits string -> Client converts each string variables into integer variable)
4. **Explain each step (for construction) or each portion (for code) in detail**
   (Show pictures and code snippets and explain why things are done that way, explain various concepts if needed [HTTP Requests, PWM etc.])
5. **(For coding) Show circuitry and explain**
6. **Show final product (output when code works, waterproofed motors etc)**

TIMELINE

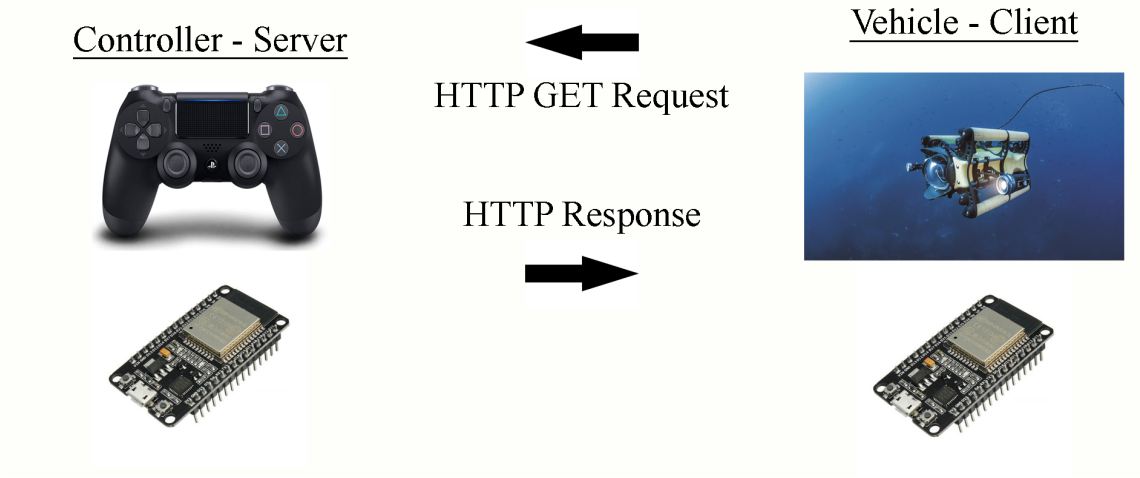
Date	Progress
17/11, T4W8	Worked on Wifi module and connecting it, worked on 3d modelling the controller
24/11, T4W10	Worked on connecting the ESP32 to one another, turning the motor on and off using the motor driver, and how to control the bot using the controller
29/11	Able to send messages across the ESP32, motors could be varied in terms of speed and direction, controller functions wired and coded
1/12	Readings from button and potentiometer could be sent across to ESP32, motors could be controlled by joystick, controller controls were mapped to an appropriate scale, got electromagnet to turn on and off
6/12	Debugging electromagnet in turning on and off with one click each, Debugging the joystick controller code, constructed the joystick controller prototype

8/12	Motor testing
14/12	3D modelling of motor holders, debugging of motor code and electromagnet code
16/12	Waterproofing of motors, soldering components together, programming the pressure sensor
20/12	Waterproofing of motors, fixing vehicle code, planning circuitry
22/12	Waterproofing of motors, started attaching motors to container, planning and fixing circuitry
12/1, T1W1	Tested the attaching and waterproofing of the vehicle, wrote out program to calibrate depth under water and adjust the motors accordingly to reach specified height

COMPONENTS

WIFI CONNECTION

Basic Structure of the Code:



Server (Controller):

1. Collect integer data from joystick module/potentiometer
2. Store it in a int variable
3. Convert variable into **string variable** using String() function
4. **Convert into a character array** string using .toCharArray()
    a. This is done as **integers cannot be sent over HTTP requests**, only data of string/char type
5. Concatenate variables into one variable
6. Send string over when **HTTP GET request is received**

Client (Vehicle):

1. Receive the string variable
2. **Split the string into 4 array items**
3. Convert each array item into a int variable for each movement/action
4. Send each variable to each component to be controlled

Controller (Server):

```
// Import required libraries (Check out
https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/
for how to set up ESP32 for Arduino IDE)
#include "WiFi.h"
#include "ESPAsyncWebServer.h"
#include <Wire.h>

// Set your access point network credentials
const char* ssid = "ESP32-Access-Point";
const char* password = "underwaterbot";

AsyncWebServer server(80);
//(Check out what Async Web Serves are:
https://randomnerdtutorials.com/esp32-client-server-wi-fi/ )

Void setup () {
  // Setting the ESP as an access point
  Serial.print("Setting AP (Access Point)…");
  // Remove the password parameter, if you want the AP (Access Point) to be open
  WiFi.softAP(ssid, password);

  IPAddress IP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(IP); //Prints the server's IP address
  // Sends the data over when HTTP Request is received (HTTP_GET)
  server.on("/potentiometer", HTTP_GET, [](AsyncWebServerRequest * request) {

    request->send_P(200, "text/plain", controlChars);
    // 200: Number of indexes available, controlChars: Variable with data
  });
    Serial.println("Network connection works");

  // Start server
  server.begin();

}
```

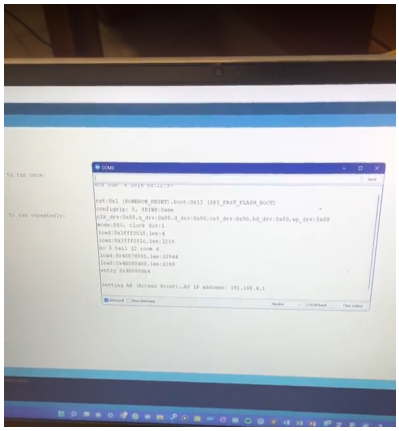
```
Void loop () {
 //Concentates integer variables into a string variable
 controls = String(potfinal) + " " + String(button) + " " + String(finalx) + " " + String(finaly);
//Changes it to a character array, this packets each data into an index with a defined separator
controls.toCharArray(controlChars, 20);

}
```

Vehicle (Client):

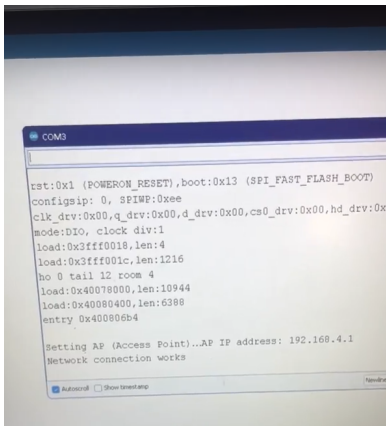
```
#include <WiFi.h>
#include <HTTPClient.h>
//include the libraries we need

const char* ssid = "ESP32-Access-Point";
const char* password = "underwaterbot";
// setting the ssid and password as variable to be used to connect to the network

unsigned long previousMillis = 0;
const long interval = 5000;
//variables we will be using the delay the GET requests

const char* serverNamepot = "http://192.168.4.1/potentiometer"; // URL to be sent over HTTP
GET Request
String getRequest;
String xvalue;
String yvalue;
String buttvalue;
String potvalue;
// setting up variables we will use

//Function that splits char array
String getValue(String data, char separator, int index)
{
  int found = 0;
  int strIndex[] = {0, -1};
  int maxIndex = data.length() - 1;

  for (int i = 0; i <= maxIndex && found <= index; i++) {
    if (data.charAt(i) == separator || i == maxIndex) {
      found++;
      strIndex[0] = strIndex[1] + 1;
      strIndex[1] = (i == maxIndex) ? i + 1 : i;
    }
```

```
  }

  return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}

//Function to send HTTP GET Request
String httpGETRequest(const char* serverName) {
  WiFiClient client;
  HTTPClient http;

  // Your Domain name with URL path or IP address with path
  http.begin(client, serverName);

  //Send HTTP GET request
  int httpResponseCode = http.GET();

  String payload = "--";

  if (httpResponseCode > 0) {
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
    payload = http.getString(); //receieves data from the HTTP GET Request
  }
  else {
    Serial.print("Error code: ");
    Serial.println(httpResponseCode);
  }
  // Free resources
  http.end();

  return payload;
}

void setup() {
  //Serial Initialisation
  Serial.begin(115200);

  //connecting to wifi
  WiFi.begin(ssid, password);
  Serial.println("Connecting");
  //If WiFi is not connected
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Waiting for connection...");
```

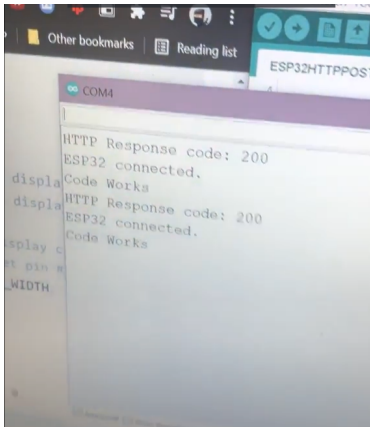
```
    }
  Serial.println("");
  Serial.print("Connected to WiFi network with IP Address:");
  Serial.println(WiFi.localIP());
}

void loop() {
  unsigned long currentMillis = millis();
    //check WiFi connection status
    if (WiFi.status() == WL_CONNECTED) {
      getRequest = httpGETRequest(serverNamepot); //calls function to send GET Request
      xvalue = getValue(getRequest, ' ', 3); //calls function to parse
      yvalue = getValue(getRequest, ' ', 2);//splits char array and gets int value
      buttvalue = getValue(getRequest, ' ', 1);//splits char array and gets int value
      potvalue = getValue(getRequest, ' ', 0);//splits char array and gets int value
      xvalue.toInt();//converts back to int
      yvalue.toInt();//converts back to int
      buttvalue.toInt();//converts back to int
      potvalue.toInt();//converts back to int
    Serial.print ("x ");
    Serial.print(xvalue);
    Serial.print(" y ");
    Serial.print(yvalue);
    Serial.println();
    Serial.println(buttvalue);
    Serial.println(potvalue);
    }
    else {
      Serial.println("WiFi Disconnected");
    }
}
```



Setting up Access Point

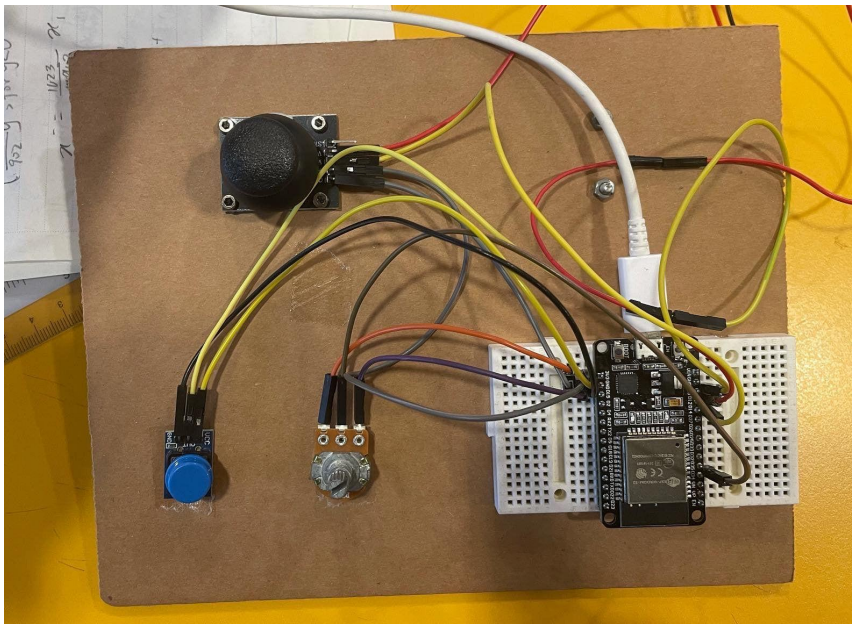


Connecting to Client



Connected, sending data

CONTROLLER



The controller could be argued as the brains of the underwater bot, hence it is crucial that the controller is programmed accurately. There were three pieces of information the controller had to transmit, one to signal in which direction the bot should move, one to signal whether to turn the electromagnet on or off, and one to determine what depth the bot should be at. To receive the input determining the direction in which the bot should move, we chose to use a joystick, as we felt that it was the most intuitive. To determine whether to turn the electromagnet on or off, we chose to use a button, as there are only two possible cases. To determine what depth to keep the bot, we chose to use a potentiometer, as it allows us to give 4095 different values for the depth, which is a great precision, more than we needed. Also, since we didn't need to adjust the depth as often, the potentiometer was a good fit as we do not need to hold the dial to keep the depth the same, unlike a joystick.

Materials needed

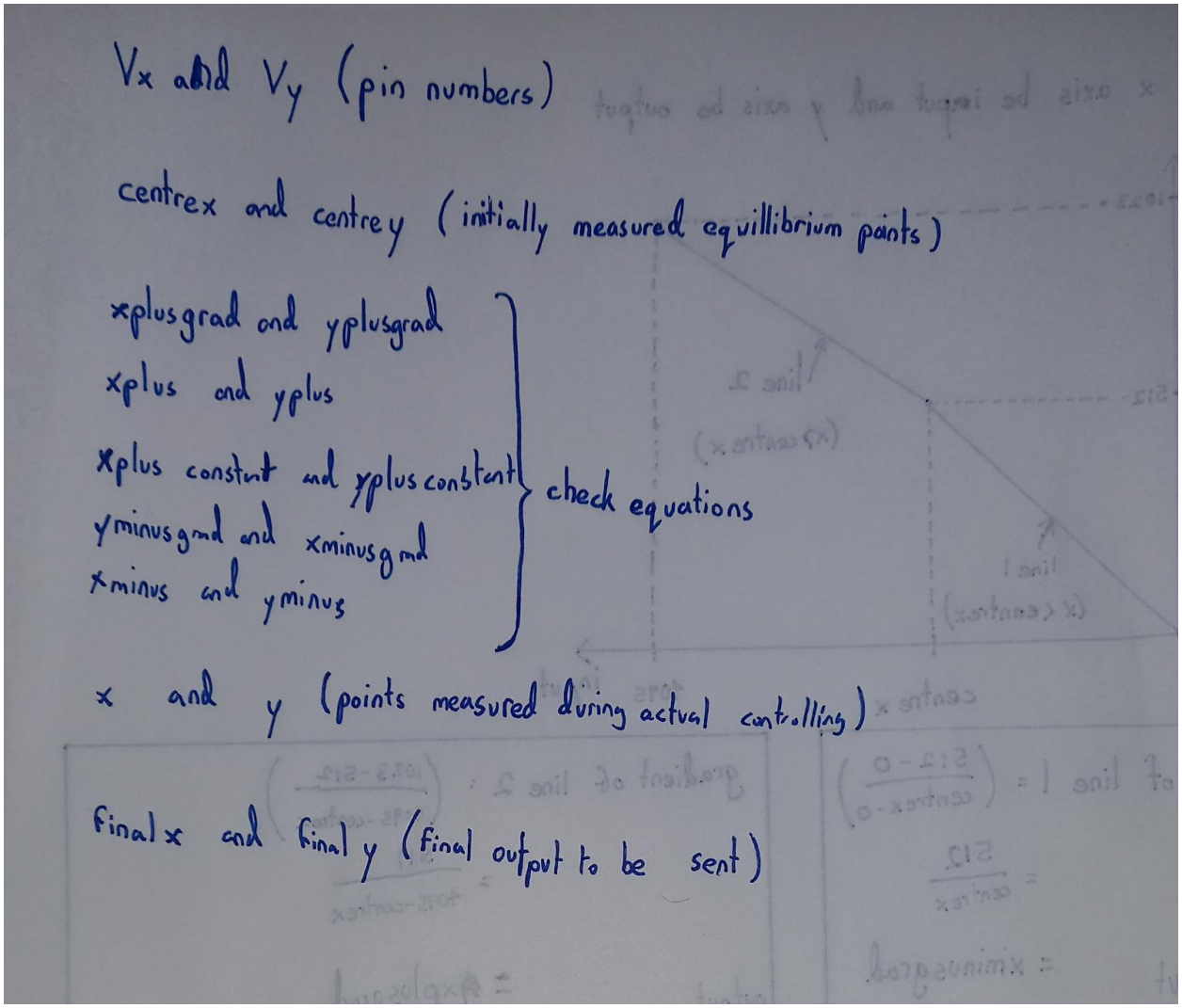
1. ESP32
2. Wires
3. Potentiometer
4. Button
5. Joystick
6. Battery and Holder
7. Cardboard (for pasting components to have a handheld controller)

Joystick Calibration

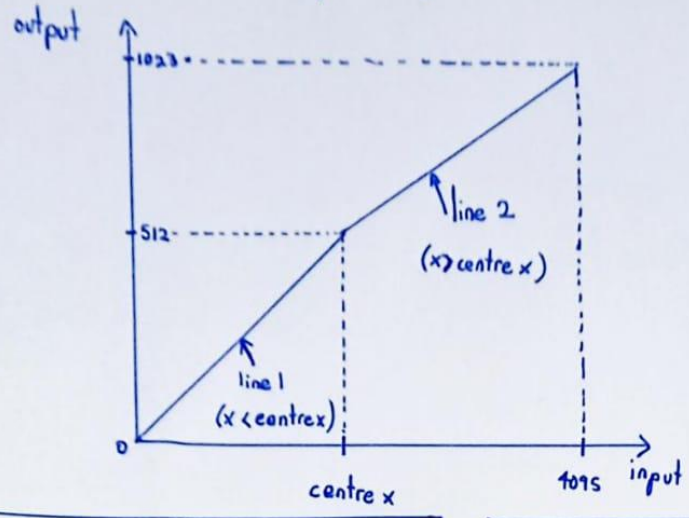
One thing that we realized during our testing phase was that in neutral positions, the joystick output tends to deviate from the expected neutral value of 2048. We initially thought this was a simple error and could just subtract the error from the recorded value. However, the problem was more complicated than we assumed. When we pushed the joystick to the maximum and

minimum, the values were 4095 and 0 respectively, meaning that the rate at which the joystick output increases or decreases is different. This problem was incredibly significant and cost us 1 to 2 sessions to figure it out entirely.

The photograph of the thinking process is attached below. If it is too hard to comprehend, read the written explanation at the bottom of the pictures while also referring to the image.



let x axis be input and y axis be output



gradient of line 1 = $\left(\frac{512-0}{centrex-0}\right)$

$= \frac{512}{centrex}$

$= xminusgrad$

output ↓ input ↓

$y = mx + c$

$y = (xminusgrad)x + c$

$c = y - (xminusgrad)_x$

substituting point $(0,0)$:

$c = 0 - (xminusgrad)(0)$

$c = 0$

$\therefore xminus = (xminusgrad)(x)$

gradient of line 2 = $\left(\frac{1023-512}{4095-centrex}\right)$

$= \frac{511}{4095-centrex}$

$= xplusgrad$

output ↓ input ↓

$y = mx + c$

$y = (xplusgrad)x + c$

$c = y - (xplusgrad)x$

substituting point $(4095, 1023)$

$c = 1023 - (xplusgrad)(4095)$

$\therefore xplus = (xplusgrad)(x) + c$

The same process applies for the y coordinate of the joystick

As we discussed earlier, the rate at which there is an increase from the equilibrium and the rate at which there is a decrease from the equilibrium is different.

We wanted to convert the joystick output, which has a range of 0 to 4095, to a range of 0 to 1023. We draw a graph of the output (values transmitted by the controller) against input (the value outputted by the joystick). The only information that we know is that the value 4095 is mapped to 1023 and the value 0 is mapped to 0. We also knew that the equilibrium value of the joystick (centrex) should be mapped to 512. We assumed that the rate of increase or decrease are constants and do not change as the recorded value deviates more and more from the equilibrium, so as to make our calculations easier.

Since the gradient of a line can be determined by dividing change in y against the change in x, the gradient of line 1 (xminusgrad) is 512.0/centrex and the gradient of line 2 (xplusgrad) is 511.0/(4095.0 - centrex). The y intercept is 0 for line 1 hence it need not be considered. However, line 2 has a y intercept. Since we know one point in line 2 (4095, 1023) and now know the gradient xplusgrad, we could substitute these values into y = mx + c and find the value of c (xplusconstant).

So now that we know both the gradient and the intercepts, we can now find the accurate value of x. If x, the recorded value of the x coordinate of the joystick, is smaller than the equilibrium value, the real value of the x coordinate of the joystick, xminus, is (xminusgrad)(x). If x is greater than the equilibrium value, the real value of the x coordinate of the joystick, xplus, is xplusgrad*x + xplusconstant. Repeat this process to find the true value of the y coordinate of the joystick.

Code (without wifi code, check out full code in the github link) with explanations in blue

```
// [For potentiometer] Setting up Variables
int potpin = 34 ;
int potvalue;
int realpot;
int potfinal;

// [For Joystick] Setting up Variables
int Vx = 35;
int centrex;
float xplusgrad;
float xplusconstant;
int xplus;
int xminus;
int x;
float xminusgrad;
int finalx;
int Vy = 32;
int centrey;
```

```
float yplusgrad;
int yplus;
int yminus;
int y;
float yminusgrad;
float yplusconstant;
int finaly;

// [For button] Setting up Variables
int buttpin = 13;
int buttstate;
int button = 0;

Put your setup code void setup, to run once:
void setup() {
   Serial.begin(115200); Just in case we need to troubleshoot and need to check the values that
are outputted

centrex = analogRead(Vx); Reading Initial value of the x component of the joystick output from a
scale of 0 to 4095

xplusgrad = 511.0/(4095.0 - centrex); Read earlier section on “Joystick Correction” to
understand why
xplusconstant = 1023.0 - xplusgrad*4095.0;
xminusgrad = 512.0/centrex;


centrey = analogRead(Vy); Reading Initial value of the y component of the joystick output from a
scale of 0 to 4095

yplusgrad = 511.0/(4095.0 - centrey); Read earlier section on “Joystick Correction” to
understand why
yminusgrad = 512.0/centrey;
yplusconstant = 1023.0 - yplusgrad*4095.0;
}

Add code in void loop if you would like to leave it continuously running
void loop() {
 // For potentiometer
 potvalue = analogRead(potpin); Reading the value of the potentiometer output from a scale of 0
to 4095
 realpot = map(potvalue, 0, 4095, 0, 1023); Converting value of potentiometer from a scale of
4095 to 1023
```

```
 potfinal = 1023 - realpot; We wanted to spin the potentiometer clockwise to increase rather than anti clockwise
 //Serial.println(String(potvalue)+" "+String(potfinal)); Check if values have any errors

 // For Joystick
 x = analogRead(Vx); Reading value of the x component of the joystick output
 y = analogRead(Vy); Reading value of the y component of the joystick output

xplus = xplusgrad*x + xplusconstant; Read earlier section on “Joystick Correction” to understand why
xminus = xminusgrad*x;
if (x >= centrex) {
  finalx = xplus;
} else {
  finalx = xminus;
}
Serial.print("x "); Check if values are correct
Serial.print(finalx);

yplus = yplusgrad*y + yplusconstant; Read earlier section on “Joystick Correction” to understand why
yminus = yminusgrad*y;
if (y >= centrey) {
  finaly = yplus;
} else {
  finaly = yminus;
}
Serial.print("  y ");Check if values are correct
Serial.println(finaly);


 // For button
 buttstate = digitalRead(buttpin); Read output of button, either 1 or 0


  if (buttstate == 1) {  If the button is pressed
    button++;  Add one to the button variable
if (button==2){
button = 0;  Covert button variable back to 0 if it reaches 2, so that the only possible numbers are 0 and 1
}
    while (buttstate == 1)  Since the arduino is very fast, sometimes one press may be registered as multiple presses as the code repeats. Using the while statement, we ensure that the arduino
```

waits for the button output to become zero before continuing to ensure that the button press is only registered as one press.

```
   {
    buttstate = digitalRead(buttpin);  Set the state of the button to its current state, so that it will become 0 when the button is released
     }

 }
 else {
 }

}
```

MOTORS

The most important considerations for the motors are:

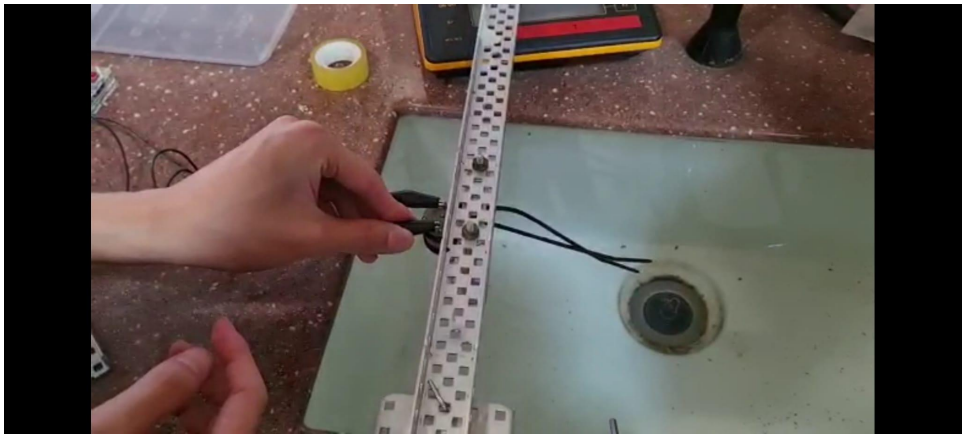
- Waterproof plz
- How much thrust does each motor provide, and is it enough (alongside upthrust of robot) to push the robot upwards?
    - As a sidenote, our robot was designed to naturally sink, so we only had to worry about the motors providing thrust upwards, and none propelling downwards
- How to attach to box? Connection needs to be waterproof also, and wires need to be sealed.

The solutions we came up with:

- Motors were first completely covered in insulating tape after wires were soldered on.
- 3D printed cases for the motors that we filled up with candle wax (paraffin btw, we melted it ourselves). Wires on the motor were also sealed in wax
  Insert pics
- Drill a hole into the sides of the box, which we put wires through and then sealed up with glue
  Insert pics

Testing the motors:

- We had this setup: (Yes this is the sink in the room, we filled it with water and tested the motors there)



Here we just wanted to see how powerful the motors were by measuring how much extra weight they added on to the weighing scale (which was originally at 0) and whether or not the motor was drawing excess current (which would cause them to heat up easily and be a safety issue) Ref doc here. Settled on a choice that would balance both strength as well as lower current that can't kill.

We had 2 sets of motors: 2 for vertical movement, the other 2 for horizontal movement. We could use weaker motors for horizontal movement as the resistive water forces are not as significant for horizontal movement compared to the weight of the robot.
Insert pic

Pseudocode:
Note: this part uses the x and y variables with respect to the ones under wifi connection, so anytime those 2 are referenced take it with context that the values are being sent from wifi.
Programming reference here

First, define the motors being used. Here we have 4 motors, and each have 2 pins that must be defined. They must be connected to PWM pins. H stands for horizontal and V stands for vertical. On the robot, we have defined a forward direction to prevent confusion.

More defining follows, pwmChannel just stands for what available channel you will dedicate to read a certain pwm output. Need 1 per motor pin. The rest will be explained at the side as necessary for definitions.

Next, the actual execution. "xzmovement" & "ymovement" are both functions that need to be defined outside of void loop(). What they do is to control the horizontal axis and vertical axis movement respectively. The first takes in 2 floats, while the second takes 1 float, all 3 from a range of 0 to 1023. The rest of the function is just processing and manipulation to program the motors proportionally to the extent the joystick and potentiometer are adjusted.

Code:
int motor1Pin1 = 13; //HRight motor forward (Right motor horizontal movement)
int motor1Pin2 = 12; //HRight motor back

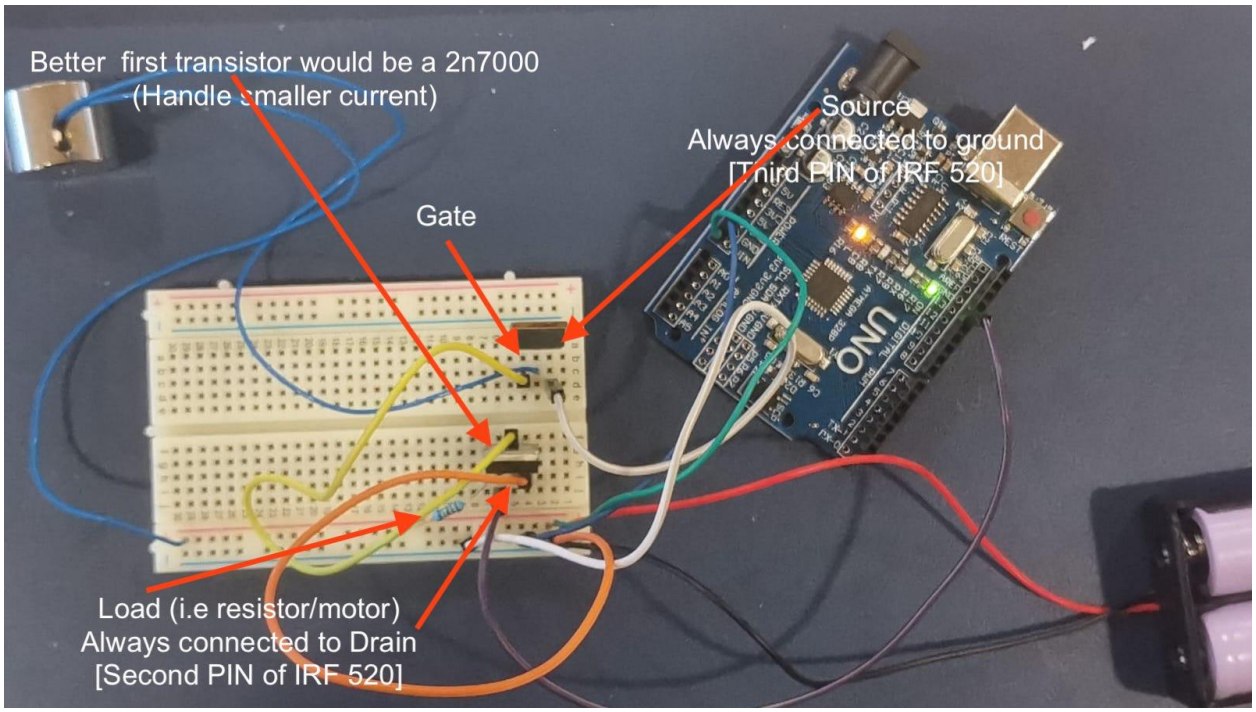
```
int motor2Pin1 = 27; //HLeft motor forward (Left motor horizontal movement)
int motor2Pin2 = 26; //HLeft motor back
int motor3Pin1 = 25; //VRight motor forward (Right motor vertical movement)
int motor3Pin2 = 33; //VRight motor back
int motor4Pin1 = 32; //VLeft motor forward (Left motor vertical movement)
int motor4Pin2 = 35; //VLeft motor back

const int freq = 30000; (Signal frequency for the setup to operate, change as necessary)
const int pwmChannel1 = 0;
const int pwmChannel2 = 1;
const int pwmChannel3 = 2;
const int pwmChannel4 = 3;
const int pwmChannel5 = 4;
const int pwmChannel6 = 5;
const int pwmChannel7 = 6;
const int pwmChannel8 = 7;
const int resolution = 10; (Duty cycle resolution is 10, which basically means your motor can
accept values from 0 to 1023, or 2^10 different values)
int DutyCycleL = 0; //Range of 0 to 1023, Left (These 3 are storage variables that we will change
later on. We have left and right as separate here so that we can control turning)
int DutyCycleR = 0; //Range of 0 to 1023, Right
int DutyCycleU = 0; //Range of 0 to 1023, Up

void setup() {
  pinMode(motor1Pin1, OUTPUT); (Just defining each motor pin to be an output)
  pinMode(motor1Pin2, OUTPUT);
  pinMode(motor2Pin1, OUTPUT);
  pinMode(motor2Pin2, OUTPUT);
  pinMode(motor3Pin1, OUTPUT);
  pinMode(motor3Pin2, OUTPUT);
  pinMode(motor4Pin1, OUTPUT);
  pinMode(motor4Pin2, OUTPUT);
  ledcSetup(pwmChannel1, freq, resolution); (Function specific to esp32, basically sets up the
function on the esp32 to program the motor)
  ledcAttachPin(motor1Pin1, pwmChannel1); (Attaches the function with specified properties to
this specific motor pin. Since they all use the same properties, so they were previously defined
with the variable above)

  ledcSetup(pwmChannel2, freq, resolution);
  ledcAttachPin(motor1Pin2, pwmChannel2);

  ledcSetup(pwmChannel3, freq, resolution);
  ledcAttachPin(motor2Pin1, pwmChannel3);
```

```
  ledcSetup(pwmChannel4, freq, resolution);
  ledcAttachPin(motor2Pin2, pwmChannel4);

  ledcSetup(pwmChannel5, freq, resolution);
  ledcAttachPin(motor3Pin1, pwmChannel5);

  ledcSetup(pwmChannel6, freq, resolution);
  ledcAttachPin(motor3Pin2, pwmChannel6);

  ledcSetup(pwmChannel7, freq, resolution);
  ledcAttachPin(motor4Pin1, pwmChannel7);

  ledcSetup(pwmChannel8, freq, resolution);
  ledcAttachPin(motor4Pin2, pwmChannel8);
}

void loop() {
  xzmovement(xvalue.toFloat(), yvalue.toFloat()); //Placeholders (the xvalue and value to float here are just for the left and right, this one isn't x and y axis. Function ".toFloat" converts the datatype from an integer to a float. This set comes from the joystick to control horizontal motors
  ymovement(potvalue.toInt()); //Placeholders (the value here is for a potentiometer, and this adjusts the strength of the vertical motors i.e. how fast do you want to go up)

}

void xzmovement(float xvalue, float yvalue) {
  /*
     Will recieve a value from 0 to 1023
     If 512 < value, move forward (So in this code, midpoint where motors do not move is at 512)
     If 512 > value, move backward
  */
  X = (xvalue / 1023.0) * 2 - 1; (We convert proportionally the PWM value into a decimal that ranges from -1 < x < 1, to make it easier to know which direction the motors will move and by how much. You can't do this with map() since map only returns integers, and will truncate fractions. While this step isn't strictly necessary, it makes life in the next few lines a lot easier)
  Z = (yvalue / 1023.0) * 2 - 1;

  Left = X - Z; (The simplest way to explain this is here, if you have any questions dm the seniors. If you still have trouble, try drawing a table and mapping out 9 different possibilities for both motors, 8 for each extreme on the joystick and 1 for the centre)
  Right = X + Z;
  if (Left > 1) { (This part is just to make sure that the values never exceed 1 and -1 respectively)
    Left = 1;
  } else if (Left < -1) {
```

```
    Left = -1;
  } else if (Right > 1) {
    Right = 1;
  } else if (Right < -1) {
    Right = -1;
  }

  DutyCycleL = Left * 1023.0; (This is converting to -1023 < x < 1023, since those are the values the motors accept)
  DutyCycleR = Right * 1023.0;
  Serial.println(String(DutyCycleL) + "  " + String(DutyCycleR));
  if (DutyCycleL >= 0) { (Checking the sign on the DutyCycle value, since how the robot moves backwards is by giving the other motor pin the value instead)
    ledcWrite(pwmChannel3, DutyCycleL);
    ledcWrite(pwmChannel4, 0);
  } else if (DutyCycleL < 0) {
    ledcWrite(pwmChannel3, 0);
    ledcWrite(pwmChannel4, abs(DutyCycleL));
  }

  if (DutyCycleR >= 0) {
    ledcWrite(pwmChannel1, DutyCycleR);
    ledcWrite(pwmChannel2, 0);
  } else if (DutyCycleR < 0) {
    ledcWrite(pwmChannel1, 0);
    ledcWrite(pwmChannel2, abs(DutyCycleR));
  }
  //delay(700);

}

void ymovement(int potvalue){
  DutyCycleU = potvalue; (Taken from controller code, basically just take this value and process it as per the other function)
  ledcWrite(pwmChannel5, DutyCycleU);  //
  ledcWrite(pwmChannel6, 0);          // Need to confirm which is up direction and which is down, placeholder values for now  (This part is incomplete, but basically same idea as above: Need to check sign of DutyCycleU by first converting to -1 < x < 1, and then applying the data to the respective motors. No need to do the X + Z & X - Z here since both vertical motors will not be used for turning [No barrel rolling], hence no need to control them independently)
  //Serial.println(DutyCycleU);
  /*
  ledcWrite(pwmChannel7, DutyCycleU);  //
  ledcWrite(pwmChannel8, 0);          //
```

```
*/
  }
```

ELECTROMAGNET



Equipment Required:

1. Arduino UNO
2. Transistor x2
    a. IRF520 [https://components101.com/mosfets/irf520-pinout-datasheet-features#:~:text=The%20IRF520%20is%20a%20Power,for%20switching%20high%20current%20loads.]
    b. 2n7000 [https://sg.element14.com/on-semiconductor/2n7000/mosfet-n-channel-200ma-60v-to/dp/9845178?gclid=EAIaIQobChMI2ZHThbHk-AIVV7eWCh0NFQTKEAAYASAAEgKiivD_BwE&mckv=slP89hIqg_dc|pcrid|563028541361|pkw|2n7000|pmt|p|slid||product||pgrid|41063249223|ptaid|kwd-297091690761|&CMP=KNC-GSG-GEN-SKU-FAI-HVHC]
3. Wires
4. Resistor
5. Electromagnet 12V

PseudoCode:

Goal of code:

Basically this code is to program a button and the magnet. While you could ignore the transistor and do the setup without, it can be pretty dangerous since the magnet used requires like 20V,

and without a transistor you're running like a lot of current in all parts of the circuit. So just use a transistor

Simple explanation:
Define necessary variables for a button (Input) and a digital pin (Output) that will work as the ON/OFF toggle for the magnet.

Read the button state
If high, then toggle magnet on
If low, then toggle off

Since Arduino reads the program several times per second, but we only want to press the button once and have the program keep doing the function (normally you have to hold the button to keep the electromagnet on, since only then the circuit is completed), we force the program to stop after reading high once, until it reads low. This allows us to not have to press the button within 1/9600th of a second to get 1 input without infinitely adding the storage variable (buttman). Then, the programme checks if the storage variable has a remainder of 1 or 0, and executes accordingly.

Circuitry (Without button):

A transistor has three pins: Gate, Drain & Source.

- The gate (first pin) is connected to the pin of the arduino/esp or the remaining component of the circuit
- The drain (second pin) is always connected to the load (i.e resistor/motor)
- The source (third pin) is always connected to the ground
- Note: The pins will vary from resistor model, so be sure to check online what are the specs before attempting not to fry it

Transistor 101: When voltage to the source pin is high, current will be allowed to flow across the gate and source pin to power whatever lies beyond the source pin. Else, current cannot flow. Its basically another button, but electrically controlled
Transistor model:

Code:

```
int buttPin = 2; (Creates a variable whose "pins" is 2)
int ledPin =  12; (Creates a variable whose "pins" is 12)
int buttState; (Defining variable)
int remainder; (Defining variable)
int buttman = 0; (Creates a variable whose "value" is 0)

void setup() {
  pinMode(ledPin, OUTPUT); (Provides voltages, HIGH or LOW (0V), read using digitalWrite)
  pinMode(buttPin, INPUT); (Returns the voltage there, HIGH or LOW, read using digitalRead)

  Serial.begin(9600);
```

```
}

void loop() {

  buttState = digitalRead(buttPin); (Returns a value of 0 or 1, 0 when LOW, 1 when HIGH. [i.e whether there is a voltage there])

  if (buttState == 1) { (when buttState “value” is 1)
    buttman=buttman+1;(Adds 1 to the “value” of buttman everytime the button is pressed)
    while (buttState == 1) (Infinite loop)

    {
     buttState = digitalRead(buttPin); (Resets buttState “value” to 0)
      }

  }
  else { (when buttState “value” is 0)
  }

remainder = buttman % 2; (Finding the remainder of the “value” buttman possesses)

  if (remainder == 1) { (when remainder is 1)
    Serial.println("Magnet is on");
    digitalWrite(ledPin,LOW); (Electromagnet switches on because the current direction is reversed)
  }
  else { (when remainder is any other value)
   Serial.println("Magnet is off");
   digitalWrite(ledPin,HIGH); (Electromagnet switches off)
  }


  }
```

PRESSURE SENSOR

***Steps to download the library:

- Open Arduino and go to **Sketch**
- Select **Include Library** and press **Manage Libraries…**
- In the search bar, search BMP 280
- Download the first search result (for me) labelled **Adafruit BMP280 Library**

Components used:

1. BME280 Pressure Sensor
2. Arduino UNO

3. Wires
4. BreadBoard
5. Plastic bag/Balloon (Anything that will inflate and deflate based on external pressure conditions)

Obejective of component and PseudoCode:
PseudoCode:

The pressure sensor will be stored in a balloon or plastic bag, which will be inflated or deflated via the external pressure conditions of the water. The reading given by the sensor will then have to be compared with a standard scale. From the change in pressure, the depth at which the machine is at can be calibrated using:

$$P = P_{\text{atm}} + \rho g h$$

Code: [Standard code that it applicable for the BME 280 model used in school too]

```
/*************************
  This is a library for the BMP280 humidity, temperature & pressure sensor
  This example shows how to take Sensor Events instead of direct readings

  Designed specifically to work with the Adafruit BMP280 Breakout
  ----> http://www.adafruit.com/products/2651

  These sensors use I2C or SPI to communicate, 2 or 4 pins are required
  to interface.

  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing products
  from Adafruit!

  Written by Limor Fried & Kevin Townsend for Adafruit Industries.
  BSD license, all text above must be included in any redistribution
 *************************/

#include <Wire.h>
#include <SPI.h>
#include <Adafruit_BMP280.h>

Adafruit_BMP280 bmp; // use I2C interface
Adafruit_Sensor *bmp_temp = bmp.getTemperatureSensor();
Adafruit_Sensor *bmp_pressure = bmp.getPressureSensor();

void setup() {
  Serial.begin(9600);
```

```
  while ( !Serial ) delay(100);   // wait for native usb
  Serial.println(F("BMP280 Sensor event test"));

  unsigned status;
  status = bmp.begin(BMP280_ADDRESS_ALT, BMP280_CHIPID);
  //status = bmp.begin();
  if (!status) {
    Serial.println(F("Could not find a valid BMP280 sensor, check wiring or "
                     "try a different address!"));
    Serial.print("SensorID was: 0x"); Serial.println(bmp.sensorID(),16);
    Serial.print("        ID of 0xFF probably means a bad address, a BMP 180 or BMP 085\n");
    Serial.print("   ID of 0x56-0x58 represents a BMP 280,\n");
    Serial.print("        ID of 0x60 represents a BME 280.\n");
    Serial.print("        ID of 0x61 represents a BME 680.\n");
    while (1) delay(10);
  }

  /* Default settings from datasheet. */
  bmp.setSampling(Adafruit_BMP280::MODE_NORMAL,     /* Operating Mode. */
                  Adafruit_BMP280::SAMPLING_X2,     /* Temp. oversampling */
                  Adafruit_BMP280::SAMPLING_X16,    /* Pressure oversampling */
                  Adafruit_BMP280::FILTER_X16,      /* Filtering. */
                  Adafruit_BMP280::STANDBY_MS_500); /* Standby time. */

  bmp_temp->printSensorDetails();
}

void loop() {
  sensors_event_t temp_event, pressure_event;
  bmp_temp->getEvent(&temp_event);
  bmp_pressure->getEvent(&pressure_event);

  Serial.print(F("Temperature = "));
  Serial.print(temp_event.temperature);
  Serial.println(" *C");

  Serial.print(F("Pressure = "));
  Serial.print(
);
  Serial.println(" hPa");

  Serial.println();
  delay(2000);
}
```

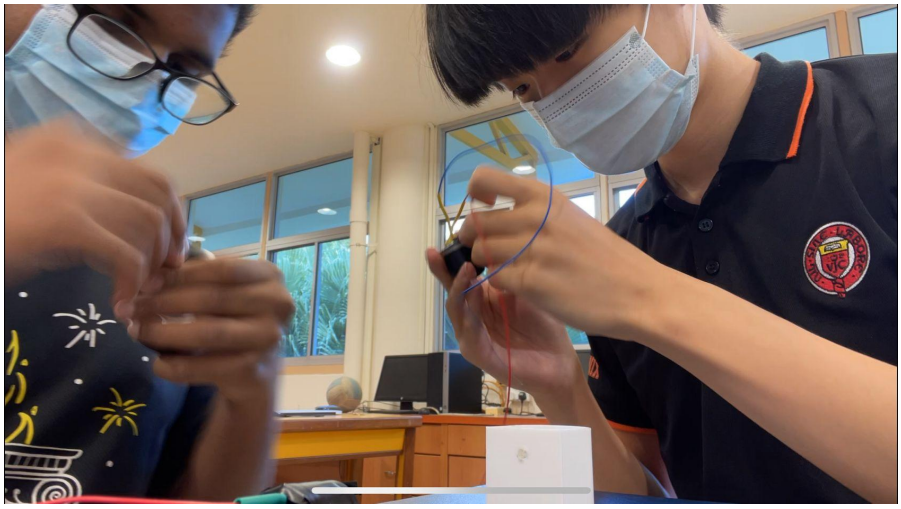
CONSTRUCTION

WATERPROOFING THE MOTORS

(Do refer to the videos of us waterproofing the motors to have a better understanding of the process: https://drive.google.com/drive/u/1/folders/1htGTWvOfFEHDlp6Q189xNIB7Q9fraSzV)

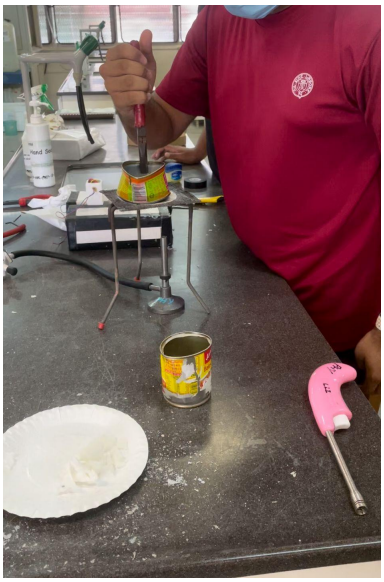
Preparation of the motors:

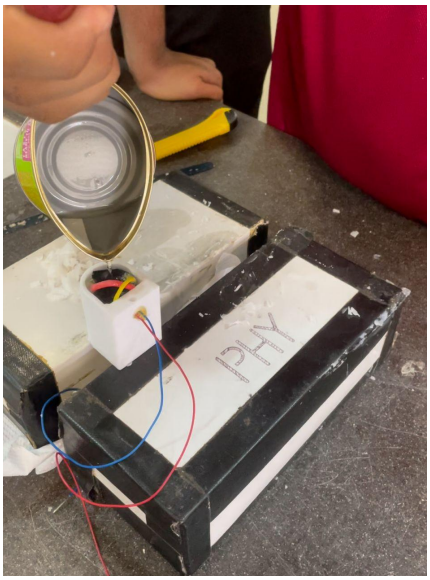
1. Solder the motors with wires and then use heat shrink the wrap around the soldered area
2. Seal the holes of the motor with electrical tape
    a. At the top part of the shaft, stick the non sticky part of the tape over it to prevent wax from entering yet allowing the shaft to still rotate
    b. Using a glue gun, hot glue the terminals of the motor
    c. Hot glue any holes in the tape to seal it
3. Start melting the wax in a pot
    a. Preferably, use a pot with a sprout to ensure proper flow of the wax when poured later
4. Apply vaseline around the motor shaft to prevent water and wax from entering



Adding wax:

5. Place the motor holder in an elevated position such that the motor shaft can poke through the shaft hole without touching the table
6. Insert the motor wires through the wire holes first and hot glue the hole
7. Cover all holes in the container (Holes for wires, shaft hole)
8. Once the wax has melted, either
    a. Place the motor into the shaft hole and start pouring wax till it reaches the brim of the motor holder
    b. Pour wax till it reaches ⅛ of the holder and then stick the motor in, ensuring the shaft goes through the shaft hole properly. Let it cool for a minute before continuing to pour the wax till it reaches the brim of the holder.
    c. Wait for the wax to cool a bit first until it is white and has a gooey texture before pouring to prevent wax from entering the motor
9. Leave it to cool for about 5 minutes





- Place layers of tissue in the surrounding area and on the bricks to prevent wax from forming on the tables or equipment used, making it hard to clean up

Materials Required:

- Wax (Wax Ring, Paraffin wax from pillar candles or tea candles)
- Electric Tape
- Pots or metal cans
- Bunsen Burner from lab/Heat Gun
- Glue gun with glue sticks
- Gloves
- Tissues
- Wire Gauze and tripod stand
- Tongs/ Pliers
- Books etc (To hold the motor up in an elevated position)

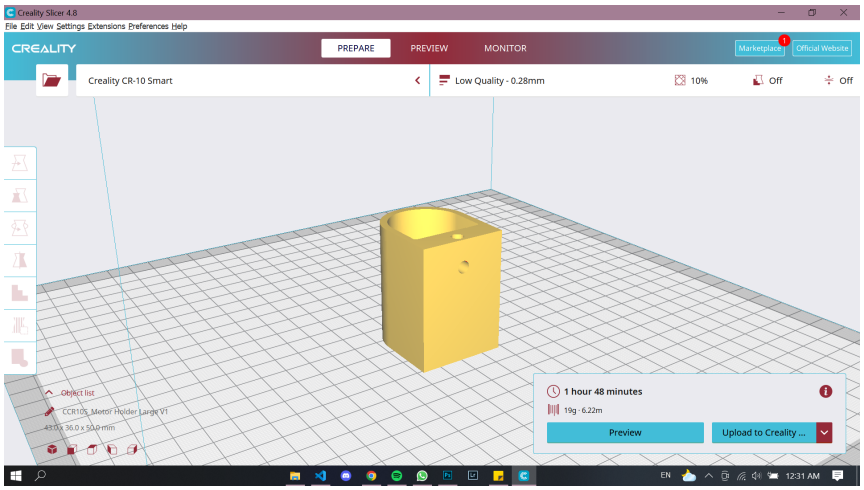
[Refer to Seaperch Manual - waterproofing the motors section if required]

MOTOR TESTING

See above

ATTACHING THE MOTORS

3D Printing:



- 3D model designed using Fusion360, Slicer used is Creality (Our 3D Printer requires the use of Creality as slicer)
- Measure dimensions of motors
- Model holder should have a clearance around the motor
- Walls need not be very thick, about 2 mm is fine
- 1mm hole is too small to be printed, 2 mm is a suitable size

- Ensure that the hole isn't too big and should be slightly undersized, can be widened by a drill bit by hand
- Cap: Joints can be printed to be the same size as the holes, can be filed to make it smaller if it doesn't fit
- Holes for wires can be bigger -- about 3-5mm
- Check the Github for the 3D models of the motor holders

Attaching the motors:

1. 3D print the motor holders and waterproof them using wax (Covered above)
2. Use a sample container/cover to estimate the hole size needed for the motor holder
3. Use a marker to cross out the hole to be made (Try not to eyeball the hole placement, use a ruler etc to measure)
4. Start drilling the hole
    a. Start with a smaller drill bit to make a smaller hole
    b. Use the drill bit needed to make the appropriate hole size needed for the motor holder
5. Insert the motor wire into the hole and ensure there are no air gaps in between the motor holder and the container
6. Seal the hole using a glue gun
7. Test for any water leakage
8. (If the position is finalised) Use epoxy and silicon sealant to add an extra layer of waterproof

PROGRESS MADE AND NEXT STEPS

<u>Completed:</u>

Wireless controller

- ☑ ~~Read from joysticks/buttons using Arduino and output to serial~~
- ☑ ~~Read from joysticks/buttons using ESP32 and output to serial~~
- ☑ ~~Test the sample code for sending messages from one esp32 to another~~
- ☑ ~~modify sample code to read from joysticks/buttons, transmit to 2nd esp32 which will output to serial~~

Main controller

- ☑ ~~Hook up motor driver, write and test code for controlling speed and direction (fwd/rev)~~
- ☑ ~~Connect electromagnet to transistor, write and test code to control it~~
- ☑ ~~Program the pressure sensor~~

Propulsion

- ☑ ~~Test and select motor~~

Construction and Assembly

- ☑ ~~Source for suitable waterproof container. Make sure everything fits~~

.

Next Steps:

1. Add on to the journal
    a. missing pictures, clarify the code structure better etc.

2. Assemble the vehicle and document it on the journal

Main controller

- [ ] Connect the pressure sensor. Find suitable library and sample code for reading sensor, test code.

Construction and Assembly

- [ ] Secure motors onto the robot frame (One is already done, but you guys will need to test if it actually is waterproof and make modifications if needed)
- [ ] Add electrical components into the robot frame and test if all components work as expected underwater

New Features

- [ ] Some way to adjust buoyancy. the boat will likely be too positively buoyant to submerge. You'll need to add weight to it.
- [ ] Some way to adjust center of gravity
- [ ] Get the pressure sensor to work with the logic required
- [ ] Learn how to use the gyro sensor and implement it in the project
- [ ] PID Controls

Resources you can refer to:

1. Apostori Link: https://www.aposteriori.com.sg/vjc/
2. Reference Document: https://docs.google.com/document/d/1vJS2Y-L5Cjpt90jwvyD31EZPmBqPyawiN7JV5KUJJAE/edit#
3. Trainer
    a. Utilise Mr Wee wisely and approach him for help regarding the robot, and discuss with Mr Kwek on when to get him to come down to help you guys work with the robot (He has about 15 hours left for this year)
    b. His Contact: 9474 3263