

## **Robotic boat for community water quality measurement**

**Mr.Theeraphat Suphakawanich Mr.Suphawit Phu-oab  
Mr.Wachiyakon sawaiamon Mr.Theeraphat Suphakawanich  
Mr.Sorawis charoentum**

*Mr.Chumpon Chareesan Miss.Netchanok Sritho Miss Tarntip Chantaranima*  
Kalasinpittayasan School, Kalasin 46000, Thailand

### **Abstract**

The project team has identified the issues of natural disasters and water pollution resulting from negligence and exploitation by water users for personal gain. Moreover, the discharge of untreated wastewater into rivers and canals exacerbates these problems. To address these challenges, they propose leveraging knowledge in science and technology, alongside automated systems, to develop a community water quality inspection robot. This robot will be remotely controlled using radio waves and will conduct surveys of distant or difficult-to-access water sources to monitor water quality.

The system comprises hardware developed with an Arduino Uno R3 Wifi ESP8266 board and software programmed in C++, which commands sensors and various devices within the system to function accurately. It facilitates the measurement of various water parameters, including Total Dissolved Solids (TDS), pH levels, and Dissolved Oxygen (DO) content. Data is stored through Google Sheets and analyzed. This setup is adept at detecting the accumulation of pollutants that may lead to water deterioration.

The accuracy of the robot's water quality measurements was validated by comparing them with water samples measured using a Vernier DOV 2-001 DO meter and a Yieryi Professional digital water quality tester (pH and TDS). The results indicate that the robot's measurements for DO ranged from 7.54 to 8.90 milligrams per liter, pH ranged from 5.08 to 5.76, and TDS ranged from 202 to 246 ppm. When compared with the Vernier DOV 2-001 DO meter and the Yieryi Professional digital water quality tester (pH and TDS), measurements for DO ranged from 7.37 to 8.73 milligrams per liter, pH ranged from 4.87 to 5.55, and TDS ranged from 184 to 228 ppm. Importantly, these measurements were consistent across all points.

### Importance and Origin







Water is important natural resources for living and economic development. Natural sources of water include water in the atmosphere (rain) surface water and groundwater are count as product from natural that human can't produce or reduce the amount that exists in nature as needed.(Pramote Maiklad.(2557).Kalasin Province has many natural sources of water. But it still faces problems with water Such as water shortages and droughts, flooding problems, and wastewater problems in natural sources of water from water users, Being neglected and invaded for personal use Draining wastewater into rivers and canals without treatment





Organizer have seen the problem and got an idea to use the knowledge in science technology and automatic system for applying and creating a robot that can analyze water in community,can be remotely control along with radio wave for explore in long distance or hard to access area. Water Quality Monitoring Robot is a set of equipment consisting of Hardware that developed by board Arduino Uno R3 ESP8266 and Software developed by C++ language that used to command sensors and components inside the system to works correctly and accurately by analyze various properties of water there are Total dissolved solids(TDS) Potential of Hydrogen ion(pH) and Dissolved oxygen(DO) then collected data and analysis by using google sheet.

### Objective

- 1.To build Robotic Boat for Community Water Quality Measurement
- 2.be able measure water quality
- 3.to monitor the occurrence of the wastewater

materials and equipment used in the development process

	
Board NodeMCU esp32	Male and female jumper wires
	
sensor tds Arduino	sensor ph arduino
	
Breadboard	sensor Do arduino

	
<p>Relay module</p>	<p>Lithium battery</p>
	
<p>Antenna</p>	<p>Motor with propeller</p>

### sensor connection

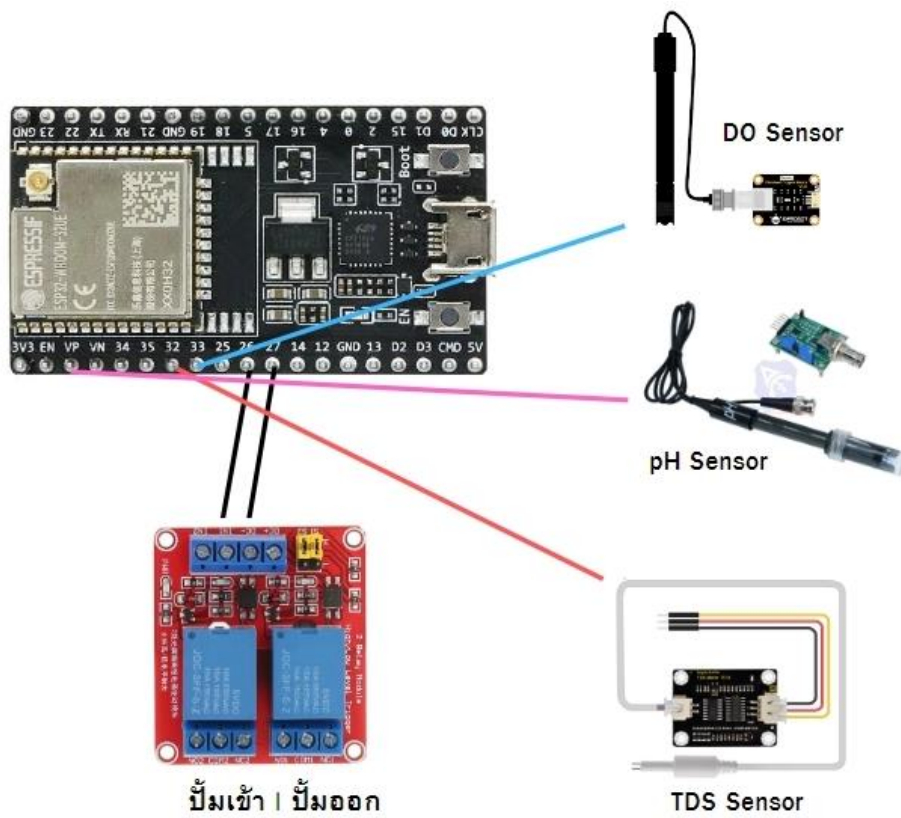


Figure 1 : Systems in Robotic boat

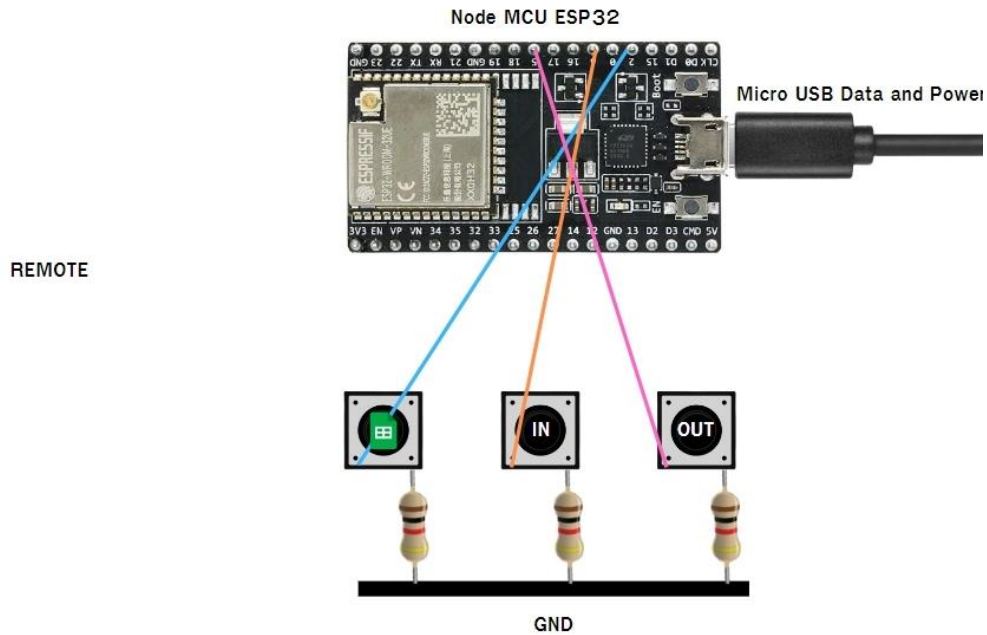


Figure 2 : System in remote

## Principles

- 1.The code begins by calling various libraries such as esp\_now, WiFi, WiFiClientSecure to utilize Wi-Fi connection, ESP-NOW communication, and sending data via HTTPS to Google Spreadsheet.
- 2.Variables are defined, including sensor pins, pump operation time, and values for processing sensor TDS in two rounds.
- 3.Functions are implemented for sending data to Google Spreadsheet using the GET request and HTTPS.
- 4.Functions are created to read values from various sensors, such as pH, Dissolved Oxygen (DO), and TDS (Total Dissolved Solids).
5. Wi-Fi Connection and Configuration for ESP8266 to Connect to Google Spreadsheet via HTTPS.
6. Utilizing ESP-NOW to Receive Data from Other Devices and Execute Received Commands such as Opening and closing the pump.

**This program operates as follows:**

1. Initiates the program and establishes a Wi-Fi connection with the specified router.
2. Reads values from various sensors such as pH, DO (Dissolved Oxygen), and TDS.
3. Waits to receive commands from other devices through ESP-NOW and executes actions based on the received commands, such as opening or closing.
4. If it receives open/close commands via ESP-NOW, the program sends signals to open or close the pump according to the received commands.
5. The program iterates and reads values from various sensors to send data to Google Spreadsheet at specified intervals.
6. If the pump is opened or closed using ESP-NOW, the program records the latest pump status in Google Spreadsheet.
7. Reading sensor values and sending data to Google Spreadsheet occurs in cycles at specified intervals and can be adjusted as needed.

**Remote(1st ESP32)**

1. **\*Libraries:\*** This code utilizes two libraries, namely `esp_now.h` and `WiFi.h`. These libraries contain functions and definitions essential for ESP-NOW usage and functions related to WiFi.
2. **\*MAC Address:\*** The variable `broadcastAddress` is an array representing the MAC address of the receiving ESP32. While the code initializes this variable, you must replace it with the actual MAC address of your receiving ESP32.
3. **\*Debounce Button:\*** There is a function called `debounce` to prevent button bouncing. This function takes the button pin number as an argument and returns the debounced state of the button.
4. **\*Variables:\*** The variables `send_rnd_val_1`, `send_rnd_val_2`, and `coolDown` are used for storing and managing the data to be sent.
5. **\*Callback Function:\*** The `OnDataSent` function serves as a callback function that executes when data is sent using ESP-NOW. This function prints the status of packet transmission.
6. **\*Setup Function:\*** The setup function manages the initialization of communication using functions in the lower section.
7. **\*Loop Function:\*** The loop function checks the status of buttons and sends data upon button presses. When buttons 1, 2, or 3 are pressed, the corresponding values are assigned to `send_rnd_val_1`, and the matching values are set in `send_Data`. Data is then

sent using ESP-NOW with `esp_now_send`. After sending the data, a cooldown period is set to prevent rapid consecutive button presses.

8. **\*Keyboard Status Management:\*** The array `key` is used to track the status of each button (pressed or released) to prevent multiple operations for the same button press. When a button is pressed, the corresponding keyboard value is set to 1 to indicate the pressed state. When the button is released, the keyboard value is set to 0.

### **System Testing\***

- Press **\*Button 1\*** to pump water into the sample collection box.
- Press **\*Button 2\*** to allow the sensor to measure the water quality.
- Please wait for each button operation to complete before pressing a new button. Do not press buttons simultaneously.
- Press **\*Button 3\*** to release water from the pump.

## Code script

```

//-----Load libraries
#include <esp_now.h>
#include <WiFi.h>
#include <WiFiClientSecure.h>

// WiFi connect timeout per AP. Increase when connecting takes longer.
const uint32_t connectTimeoutMs = 10000;
////////// TDS Sensor //////////
#define TdsSensorPin 35
#define VREF 1 // analog reference voltage(Volt) of the ADC
#define SCOUNT 30 // sum of sample point
int analogBuffer[SCOUNT]; // store the analog value in the array, read from ADC
int analogBufferTemp[SCOUNT];
int analogBufferIndex = 0, copyIndex = 0;
float averageVoltage = 0, tdsValue = 0, temperature = 25;
//-----
#define INTERVAL_TIME_PUMP_1 10000 // 10วินาที
#define INTERVAL_TIME_PUMP_2 1500 // 1.5วินาที
#define Pump_1 26
#define Pump_2 27
//-----Define variables to store incoming readings
int receive_Button;
int receive_SendSpreadsheets;
int Memory_data 1;
unsigned long time_1 = 0;
unsigned long time_2 = 0;
#define DO_PIN 33
void sendData();

#define VREF 3300 //VREF (mv)
#define ADC_RES 4096 //ADC Resolution

//Single-point calibration Mode=0
//Two-point calibration Mode=1
#define TWO_POINT_CALIBRATION 0

#define pHPin 36
float analog_pH_value = 0;
float pH_Value;

#define READ_TEMP (25) //Current water temperature °C, Or temperature sensor function

//Single point calibration needs to be filled CAL1_V and CAL1_T
#define CAL1_V (1600) //mv
#define CAL1_T (25) //°C
//Two-point calibration needs to be filled CAL2_V and CAL2_T
//CAL1 High temperature point, CAL2 Low temperature point
#define CAL2_V (1300) //mv
#define CAL2_T (15) //°C

const uint16_t DO_Table[41] = {
  14460, 14220, 13820, 13440, 13090, 12740, 12420, 12110, 11810, 11530,
  11260, 11010, 10770, 10530, 10300, 10080, 9860, 9660, 9460, 9270,
  9080, 8900, 8730, 8570, 8410, 8250, 8110, 7960, 7820, 7690,
  7560, 7430, 7300, 7180, 7070, 6950, 6840, 6730, 6630, 6530, 6410};

uint8_t Temperature;
uint16_t ADC_Raw;
uint16_t ADC_Voltage;
uint16_t DO;
String DO_Value;

int16_t readDO(uint32_t voltage_mv, uint8_t temperature_c) {
  #if TWO_POINT_CALIBRATION == 0
    uint16_t V_saturation = (uint32_t)CAL1_V + (uint32_t)35 * temperature_c - (uint32_t)CAL1_T * 35;
    int16_t doValue = (voltage_mv * DO_Table[temperature_c] / V_saturation);
    return min(doValue, 0, 16000, 0, 9000);
  #endif
}

```

Figure 3: Receiver Script 1

```

//-----
#if TWO_POINT_CALIBRATION == 0
uint16_t V_saturation = (uint32_t)CAL1_V + (uint32_t)35 * temperature_c - (uint32_t)CAL1_T * 35;
int16_t doValue = (voltage_mv * DO_Table[temperature_c] / V_saturation);
return map(doValue, 0, 16000, 0, 9000);
#else
uint16_t V_saturation = (int16_t)((int8_t)temperature_c - CAL2_T) * ((uint16_t)CAL1_V - CAL2_V) / ((uint8_t)CAL1_T - CAL2_T) + CAL2_V;
int16_t doValue = (voltage_mv * DO_Table[temperature_c] / V_saturation);
return map(doValue, 0, 16000, 0, 9000);
#endif
}

String t;
#define ON_Board_LED 2 //--> Defining an On Board LED, used for indicators when the process of connecting to a wifi router
//
//-----SSID dan Password wifi mu gan.
const char* ssid = "CE-ESL"; //--> Nama Wifi / SSID.
const char* password = "ce5lonly"; //--> Password wifi .
//-----
//-----Host & httpsPort
const char* host = "script.google.com";
const int httpsPort = 443;
//-----
// Initialize DHT sensor.
WiFiClientSecure client; //--> Create a WiFiClientSecure object.

// Timers auxiliar variables
long now = millis();
long lastMeasure = 0;

String GAS_ID = "AKfycbx4870amh_4hwVnHGy-LieehsfVvMIUm_MxEe7_RuMS3FiJzEaq6Qg56mEegpOQ78fRLg"; //--> spreadsheet script ID
//-----
//-----Structure example to receive data
// Must match the sender structure
typedef struct struct_message {
    int rnd_1;
} struct_message;

struct_message receive_Data; //--> Create a struct_message to receive data.
//-----
//+++++ Callback when data is received
void sendData(String value,float value2, float value3) {
    Serial.println("=====");
    Serial.print("connecting to ");
    Serial.println(host);

    //-----Connect to Google host
    if (!client.connect(host, httpsPort)) {
        Serial.println("connection failed");
        return;
    }
    //-----
    //-----Proses dan kirim data

    String string_DO = value;
    float string_ph = value2;
    float string_tds = value3;
    String url = "/macros/s/" + GAS_ID + "/exec?DO_Value=" + string_DO + "&ph_Value="+string_ph+ "&tds_Value="+string_tds; // 2 variables
    Serial.print("requesting URL: ");
    Serial.println(url);

    client.print(String("GET ") + url + " HTTP/1.1\r\n" +
        "Host: " + host + "\r\n" +
        "User-Agent: BuildFailureDetectorESP8266\r\n" +
        "Connection: close\r\n\r\n");
}

```

Figure 4: Receiver Script 2



```

//-----
digitalWrite(ON_Board_LED, HIGH); //--> Turn off the On Board LED when it is connected to the wifi router.
//-----If successfully connected to the wifi router, the IP Address that will be visited is displayed in the serial monitor
Serial.println("");
Serial.print("Successfully connected to : ");
Serial.println(ssid);
Serial.print("IP address: ");
Serial.println(WiFi.localIP());
Serial.println();
//-----

client.setInsecure();   WiFi.begin(ssid, password); //--> Connect to your WiFi router
Serial.println("");

pinMode(ON_Board_LED, OUTPUT); //--> On Board LED port Direction output
digitalWrite(ON_Board_LED, HIGH); //-->

//-----Wait for connection
Serial.print("Connecting");
while (WiFi.status() != WL_CONNECTED) {
  Serial.print(".");
  //-----Make the On Board Flashing LED on the process of connecting to the wifi router.
  digitalWrite(ON_Board_LED, LOW);
  delay(250);
  digitalWrite(ON_Board_LED, HIGH);
  delay(250);
  //-----
}
//-----
digitalWrite(ON_Board_LED, HIGH); //--> Turn off the On Board LED when it is connected to the wifi router.
//-----If successfully connected to the wifi router, the IP Address that will be visited is displayed in the serial monitor
Serial.println("");
Serial.print("Successfully connected to : ");
Serial.println(ssid);
Serial.print("IP address: ");
Serial.println(WiFi.localIP());
Serial.println();
//-----

client.setInsecure();
}

void pH_Sensor() {
  analog_pH_value = analogRead(pHPin);
  float voltage = analog_pH_value * (3.3 / 4095.0);
  pH_Value = (3.3 * voltage);
}

void setup() {
  Serial.begin(115200);
  pinMode(Pump_1, OUTPUT);
  pinMode(Pump_2, OUTPUT);
  pinMode(TdsSensorPin, INPUT);
  pinMode(pHPin, INPUT);

  // Connect to WiFi in STA mode
  WiFi.mode(WIFI_STA);
  //-----Init ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }
  //-----

  esp_now_register_recv_cb(OnDataRecv); //--> Register for a callback
}

void DO_Sensor(){
  Temperaturet = (uint8_t)READ_TEMP;
  ADC_Raw = analogRead(DO_PIN);
}

```

Figure 5: Receiver Script 3

```

DO_Value = String(readDO(ADC_Voltage, Temperatur));
}

int getMedianum(int bArray[], int iFilterLen)
{
  int bTab[iFilterLen];
  for (byte i = 0; i < iFilterLen; i++)
    bTab[i] = bArray[i];
  int i, j, bTemp;
  for (j = 0; j < iFilterLen - 1; j++)
  {
    for (i = 0; i < iFilterLen - j - 1; i++)
    {
      if (bTab[i] > bTab[i + 1])
      {
        bTemp = bTab[i];
        bTab[i] = bTab[i + 1];
        bTab[i + 1] = bTemp;
      }
    }
  }
  if ((iFilterLen & 1) > 0)
    bTemp = bTab[(iFilterLen - 1) / 2];
  else
    bTemp = (bTab[iFilterLen / 2] + bTab[iFilterLen / 2 - 1]) / 2;
  return bTemp;
}

void loop() {
  static unsigned long analogSampleTimepoint = millis();
  if (millis() - analogSampleTimepoint > 400) //every 400 milliseconds, read the analog value from the ADC
  {
    analogSampleTimepoint = millis();
    analogBuffer[analogBufferIndex] = analogRead(tdsSensorPin); //read the analog value and store into the buffer
    analogBufferIndex++;
    if (analogBufferIndex == SCOUNT)
      analogBufferIndex = 0;
  }
  static unsigned long printTimepoint = millis();
  if (millis() - printTimepoint > 800)
  {
    printTimepoint = millis();
    for (copyIndex = 0; copyIndex < SCOUNT; copyIndex++)
      analogBufferTemp[copyIndex] = analogBuffer[copyIndex];
    averageVoltage = getMedianum(analogBufferTemp, SCOUNT) * (float)VREF / 1024.0; // read the analog value more stable by the median filtering algorithm, and convert to voltage value
    float compensationCoefficient = 1.0 + 0.02 * (temperature - 25.0); //temperature compensation formula: fFinalResult(25°C) = fFinalResult(current)/(1.0+0.02*(fTP-25.0));
    float compensationVolatge = averageVoltage / compensationCoefficient; //temperature compensation
    tdsValue = (133.42 * compensationVolatge * compensationVolatge * compensationVolatge - 255.86 * compensationVolatge + 857.39 * compensationVolatge) * 0.5 * 0.00000001;
  }
  DO_Sensor();
  pH_Sensor();

  if(receive_Button == 1){
    delay(INTERVAL_TIME_PUMP_1);
    digitalWrite(Pump_1,LOW);
  }
  else if(receive_Button == 2){
    delay(INTERVAL_TIME_PUMP_2);
    digitalWrite(Pump_2,LOW);
  }
  else if(receive_Button == 3){
    WiFiClientSetup();
    float tds_Value = tdsValue;
    sendData(DO_Value,pH_Value,tds_Value);
    ESP.restart();
  }
}

```

Figure 6 : Receiver Script 4

```

DO_Value = String(readDO(ADC_Voltage, Temperature));
}

int getMedianum(int bArray[], int iFilterLen)
{
  int bTab[iFilterLen];
  for (byte i = 0; i < iFilterLen; i++)
  {
    bTab[i] = bArray[i];
  }
  int i, j, bTemp;
  for (j = 0; j < iFilterLen - 1; j++)
  {
    for (i = 0; i < iFilterLen - j - 1; i++)
    {
      if (bTab[i] > bTab[i + 1])
      {
        bTemp = bTab[i];
        bTab[i] = bTab[i + 1];
        bTab[i + 1] = bTemp;
      }
    }
  }
  if ((iFilterLen & 1) > 0)
  {
    bTemp = bTab[(iFilterLen - 1) / 2];
  }
  else
  {
    bTemp = (bTab[iFilterLen / 2] + bTab[iFilterLen / 2 - 1]) / 2;
  }
  return bTemp;
}

void loop() {
  static unsigned long analogSampleTimepoint = millis();
  if (millis() - analogSampleTimepoint > 480) //every 480 milliseconds, read the analog value from the ADC
  {
    analogSampleTimepoint = millis();
    analogBuffer[analogBufferIndex] = analogRead(tdsSensorPin); //read the analog value and store into the buffer
    analogBufferIndex++;
    if (analogBufferIndex == SCOUNT)
      analogBufferIndex = 0;
  }
  static unsigned long printTimepoint = millis();
  if (millis() - printTimepoint > 8000)
  {
    printTimepoint = millis();
    for (copyIndex = 0; copyIndex < SCOUNT; copyIndex++)
    {
      analogBufferTemp[copyIndex] = analogBuffer[copyIndex];
    }
    averageVoltage = getMedianum(analogBufferTemp, SCOUNT) * (float)VREF / 1024.0; // read the analog value more stable by the median filtering algorithm, and convert to voltage value
    float compensationCoefficient = 1.0 + 0.02 * (temperature - 25.0); //temperature compensation formula: fFinalResult(25°C) = fFinalResult(current)/(1.0+0.02*(fTP-25.0));
    float compensationVolatge = averageVoltage / compensationCoefficient; //temperature compensation
    float tdsValue = (133.42 * compensationVolatge * compensationVolatge * compensationVolatge - 255.86 * compensationVolatge * compensationVolatge + 857.39 * compensationVolatge) * 0.5 * 0.00000001;
  }
  DO_Sensor();
  pH_Sensor();

  if(receive_Button == 1){
    delay(INTERVAL_TIME_PUMP_1);
    digitalWrite(Pump_1,LOW);
  }
  else if(receive_Button == 2){
    delay(INTERVAL_TIME_PUMP_2);
    digitalWrite(Pump_2,LOW);
  }
  else if(receive_Button == 3){
    WiFiClientSetup();
    float tds_Value = tdsValue;
    sendData(DO_Value,pH_Value,tds_Value);
    ESP.restart();
  }
}
}

```

Figure 7 : Receiver Script 5

```

    client.setInsecure();
}

void pH_Sensor() {
    analog_pH_value = analogRead(pHPin);
    float voltage = analog_pH_value * (3.3 / 4095.0);
    pH_value = (3.3 * voltage);
}

void setup() {
    Serial.begin(115200);
    pinMode(Pump_1,OUTPUT);
    pinMode(Pump_2,OUTPUT);
    pinMode(TdsSensorPin, INPUT);
    pinMode(pHPin,INPUT);

    // Connect to WiFi in STA mode
    WiFi.mode(WIFI_STA);
    //-----Init ESP-NOW
    if (esp_now_init() != ESP_OK) {
        Serial.println("Error initializing ESP-NOW");
        return;
    }
    //-----

    esp_now_register_recv_cb(OnDataRecv); //--> Register for a callback
}

void DO_Sensor(){
    Temperature_t = (uint8_t)READ_TEMP;
    ADC_Raw = analogRead(DO_PIN);
    ADC_Voltage = uint32_t(VREF) * ADC_Raw / ADC_RES;

    DO_Value = String(readDO(ADC_Voltage, Temperature_t));
}

int getMedianNum(int bArray[], int iFilterLen)
{
    int bTab[iFilterLen];
    for (byte i = 0; i < iFilterLen; i++)
        bTab[i] = bArray[i];
    int i, j, bTemp;
    for (j = 0; j < iFilterLen - 1; j++)
    {
        for (i = 0; i < iFilterLen - j - 1; i++)
        {
            if (bTab[i] > bTab[i + 1])
            {
                bTemp = bTab[i];
                bTab[i] = bTab[i + 1];
                bTab[i + 1] = bTemp;
            }
        }
    }
    if ((iFilterLen & 1) > 0)
        bTemp = bTab[(iFilterLen - 1) / 2];
    else
        bTemp = (bTab[iFilterLen / 2] + bTab[iFilterLen / 2 - 1]) / 2;
    return bTemp;
}

void loop() {
    static unsigned long analogSampleTimepoint = millis();
    if (millis() - analogSampleTimepoint > 400) //every 400 milliseconds, read the analog value from the ADC
    {
        analogSampleTimepoint = millis();
        analogBuffer[analogBufferIndex] = analogRead(TdsSensorPin); //read the analog value and store into the buffer
    }
}

```

Figure 8 : Receiver Script 6

## Remote control scripts

```

#include <esp_now.h>
#include <WiFi.h>

uint8_t broadcastAddress[] = {0xC0, 0x49, 0xEF, 0xD0, 0x30, 0x30}; //--> REPLACE WITH THE MAC Address of your receiver / ESP32 Receiver.

//-----Variables to accommodate the data to be sent.
int send_rnd_val_1;
int send_rnd_val_2;
int coolDown;
//-----Debounce button -----
//int runn = 0;
int Button[4] = {2,4,5,18};
//bool reading[4];
//bool buttonstate[4] = {LOW,LOW,LOW,LOW};
//bool lastbuttonstate[4] = {LOW,LOW,LOW,LOW};
//unsigned long long int lastDebounceTime[4];
//-----initfunctions

int debounce(int pin) {
  const int delay_ms = 50; // debounce time in milliseconds
  static int last_readings[4] = {0, 0, 0, 0}; // last 4 readings of each button
  int reading = digitalRead(Button[pin]); // read the current state of the button
  if (reading != last_readings[pin]) { // if the current state is different from the last 4 readings
    delay(delay_ms); // wait for the debounce time
    reading = digitalRead(Button[pin]); // read the current state again
    if (reading != last_readings[pin]) { // if it's still different
      last_readings[pin] = reading; // update the last readings
      return reading; // return the current state
    }
  }
  return -1; // return -1 if the state hasn't changed or has been debounced
}

//-----

String success;

//-----Structure Pump Relay_1 to send data
// Must match the receiver structure
typedef struct struct_Pump {
  int rnd_1;
} struct_Pump;

struct_Pump send_Data;

//+++++++ Callback when data is sent
void onDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
  if (status == 0){
    success = "Switch Pump Success :";
  }
  else{
    success = "Switch Pump Fail :(";
  }
  Serial.println(">>>>");
}

void setup() {
  Serial.begin(115200);
  for(int i=0;i<3;i++){
    pinMode(Button[i],INPUT);
  }
  WiFi.mode(WIFI_STA); //--> Set device as a Wi-Fi Station.

  //-----Init ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
  }
}

```

Figure 9 : Sender Script 1

```

if (esp_now_init() != ESP_OK) {
  Serial.println("Error initializing ESP-NOW");
  return;
}
//-----

//-----Once ESPNow is successfully Init, we will register for Send CB to
// get the status of Trasnmitted packet
esp_now_register_send_cb(OnDataSent);
//-----

//-----Register peer
esp_now_peer_info_t peerInfo;
memcpy(peerInfo.peer_addr, broadcastAddress, 6);
peerInfo.channel = 0;
peerInfo.encrypt = false;
//-----

//-----Add peer
if (esp_now_add_peer(&peerInfo) != ESP_OK){
  Serial.println("Failed to add peer");
  return;
}
//-----
}

int key[4]={0,0,0,0};
void loop() {
  debounce(0);
  debounce(1);
  debounce(2);
  debounce(3);
  //////////////// Button 1 - Pump 1 ////////////////
  if(digitalRead(Button[0]) == HIGH && key[0]==0){
    if(millis() - coolDown > 80){
      //-----Set values to send Pump-1
      send_rnd_val_1 = 1;
      send_Data.rnd_1 = send_rnd_val_1;
      //-----
      Serial.println();
      Serial.print(">>>>> ");
      Serial.println("Send BUTTON_1");
      esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &send_Data, sizeof(send_Data));
      if (result == ESP_OK) {
        Serial.println("Sent with success");
      }
      else {
        Serial.println("Error sending the data");
      }
      coolDown = millis();
      key[0]=1;
    }
  }
  //////////////// Button 2 - Pump 2 ////////////////
  if(digitalRead(Button[1]) == HIGH && key[1]==0){
    if(millis() - coolDown > 80){
      //-----Set values to send Pump-1
      send_rnd_val_1 = 2;
      send_Data.rnd_1 = send_rnd_val_1;
      //-----
      Serial.println();
      Serial.print(">>>>> ");
      Serial.println("Send BUTTON_2");
      esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &send_Data, sizeof(send_Data));
      if (result == ESP_OK) {
        Serial.println("Sent with success");
      }
      else {
        Serial.println("Error sending the data");
      }
    }
  }
}

```

Figure 10 : Sender Script 2

```

    }
    else {
        Serial.println("Error sending the data");
    }
    coolDown = millis();
    key[0]=1;
}
}
////////////////////////////////////////////////// Button 2 - Pump_2 ////////////////////////////////////////
if(digitalRead(Button[1]) == HIGH && key[1]==0){
    if(millis() - coolDown > 80){
        //-----Set values to send Pump-1
        send_rnd_val_1 = 2;
        send_Data.rnd_1 = send_rnd_val_1;
        //-----
        Serial.println();
        Serial.print(">>>>> ");
        Serial.println("Send BUTTON_2");
        esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &send_Data, sizeof(send_Data));
        if (result == ESP_OK) {
            Serial.println("Sent with success");
        }
        else {
            Serial.println("Error sending the data");
        }
    }
    key[1]=1;
    coolDown = millis();
}
}
////////////////////////////////////////////////// Button 3 - Google Sheet ////////////////////////////////////////
if(digitalRead(Button[2]) == HIGH && key[2]==0){
    if(millis() - coolDown > 70){
        //-----Set values to send Pump-1
        send_rnd_val_1 = 3;
        send_Data.rnd_1 = send_rnd_val_1;
        //-----
        Serial.println("Button 3 : Google Sheets");
        esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &send_Data, sizeof(send_Data));
        key[2]=1;
    }
    coolDown = millis();
}
}
////////////////////////////////////////////////// Button 4 - Line Notify ////////////////////////////////////////
if(digitalRead(Button[3]) == HIGH && key[3]==0){
    if(millis() - coolDown > 70){
        //-----Set values to send Pump-1
        send_rnd_val_1 = 4;
        send_Data.rnd_1 = send_rnd_val_1;
        //-----
        Serial.println("Button 4 : LINE Notify");
        esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &send_Data, sizeof(send_Data));
        key[3]=1;
    }
    coolDown = millis();
}
}
//----- Release key Button -----//
if(digitalRead(Button[0]) == LOW && key[0]==1){
    key[0]=0;
}
if(digitalRead(Button[1]) == LOW && key[1]==1){
    key[1]=0;
}
if(digitalRead(Button[2]) == LOW && key[2]==1){
    key[2]=0;
}
if(digitalRead(Button[3]) == LOW && key[3]==1){
    key[3]=0;
}
}
//+++++

```

Figure 11 : Sender Script 3



**Image of invention/creation**

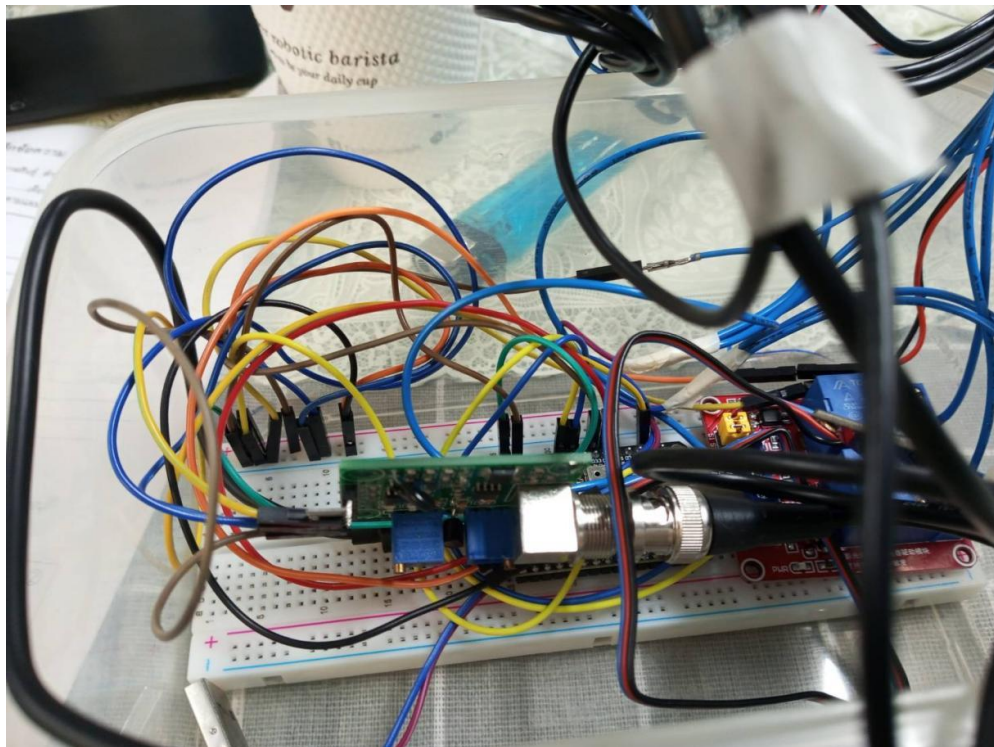


Figure 12 : Process of making Remote

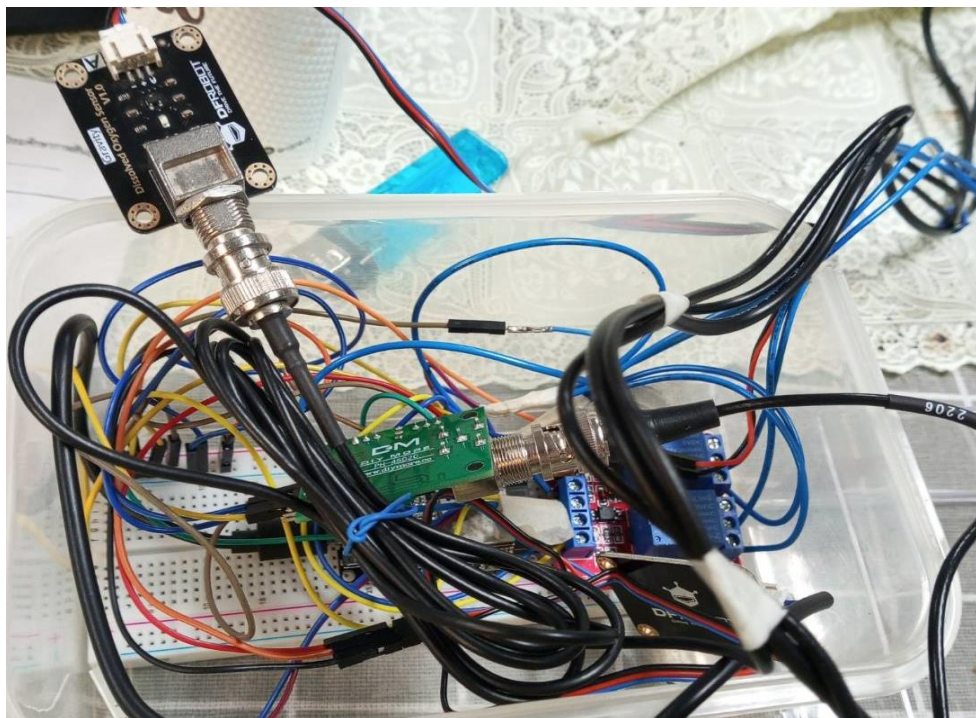


Figure 13: Process of making Remote 2



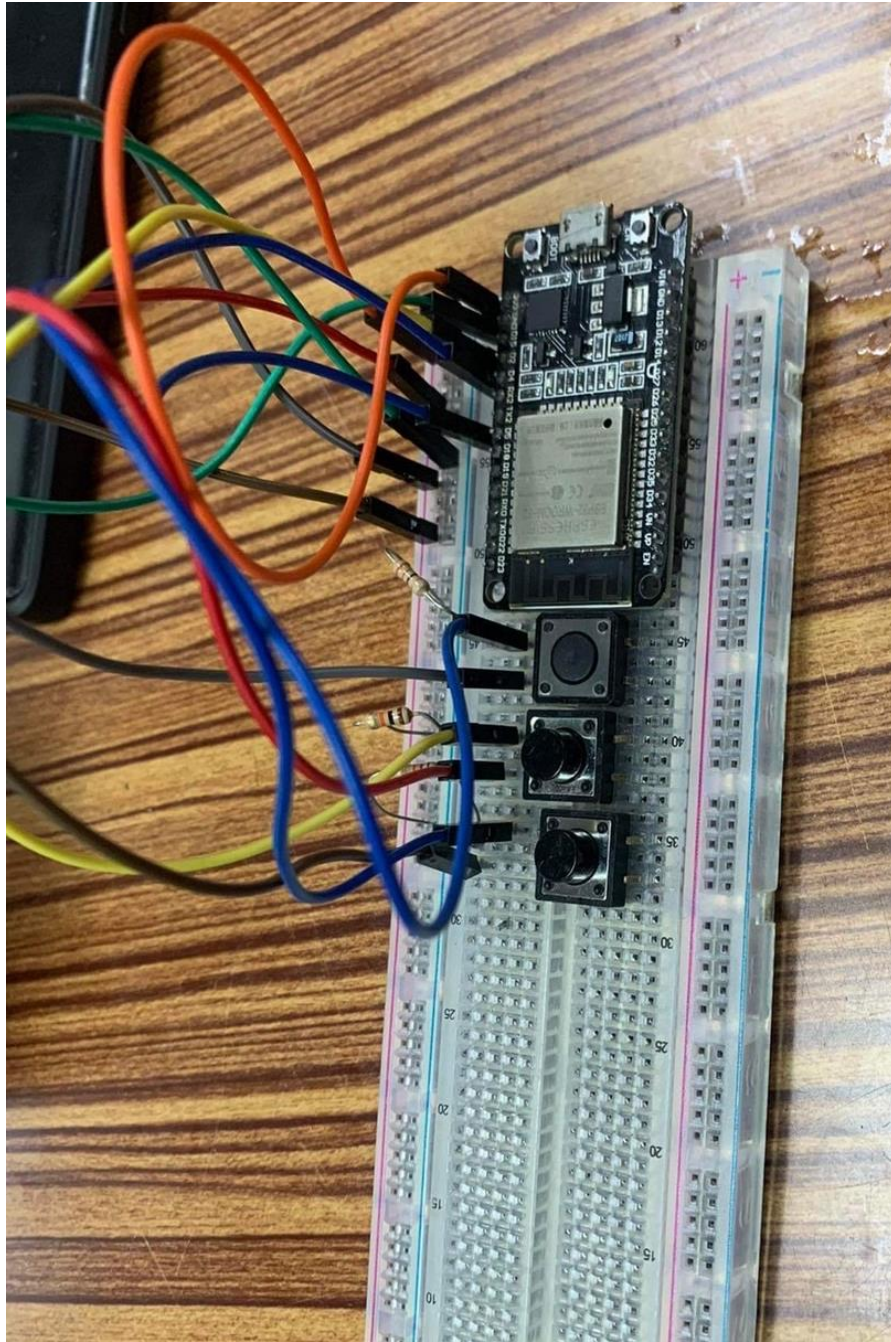


Figure 14 : Process of making Remote 3

**Design robotic boat by using AutoCAD**

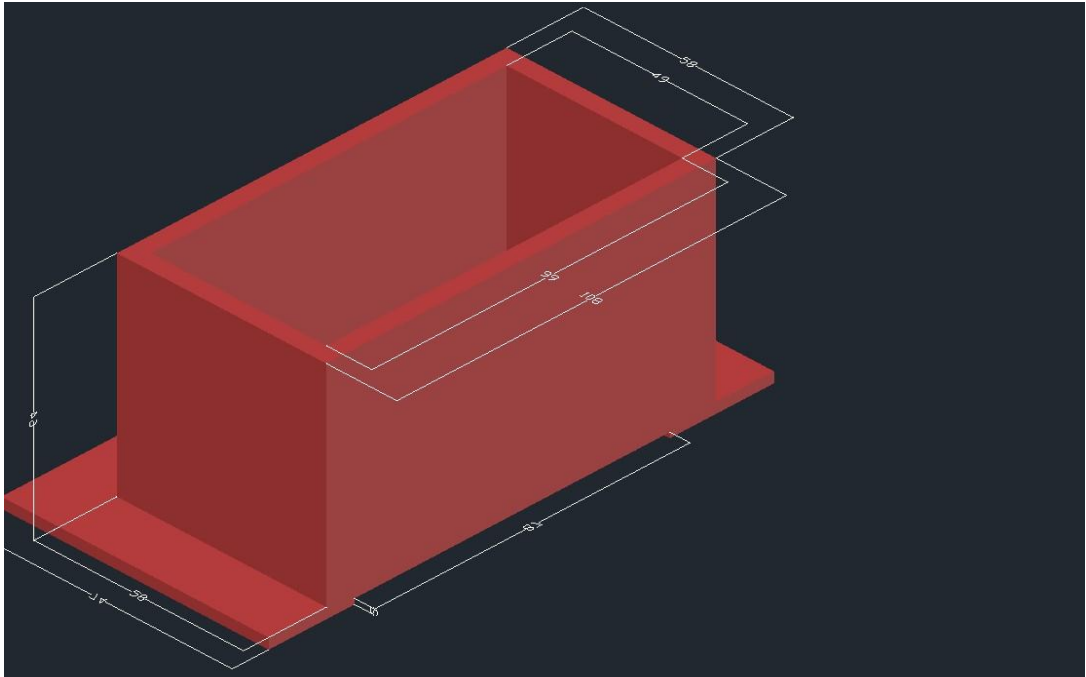


Figure 15 : Process of desining robotic boat by AutoCAD 1

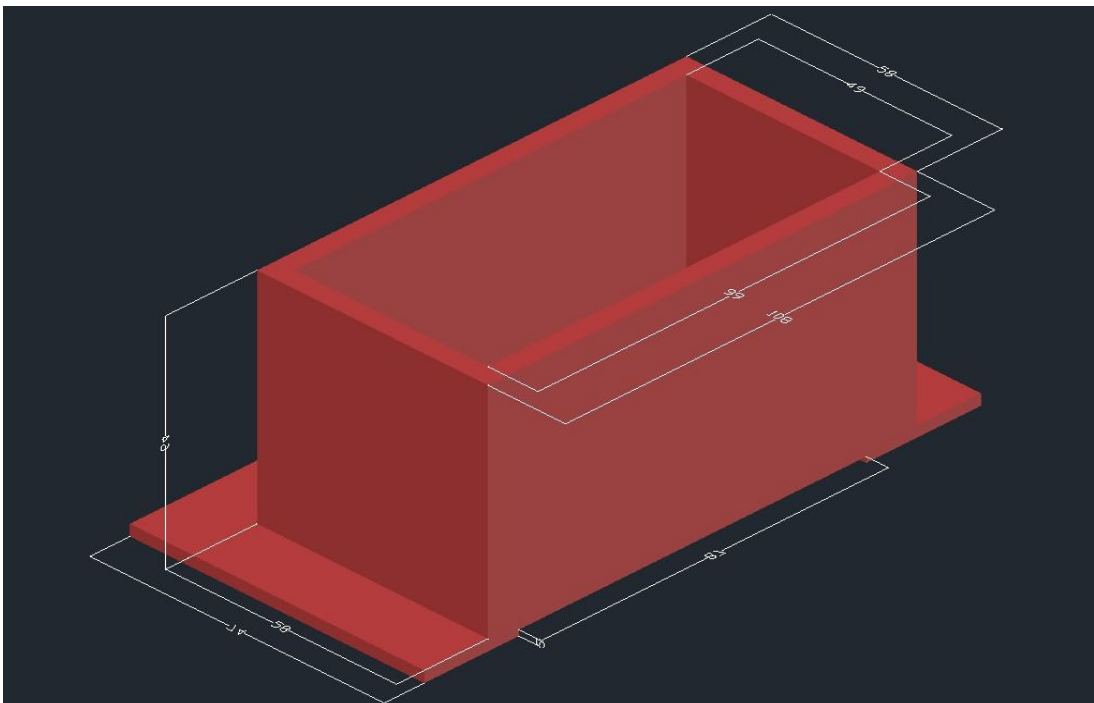


Figure 16 : Process of desining robotic boat by AutoCAD 2

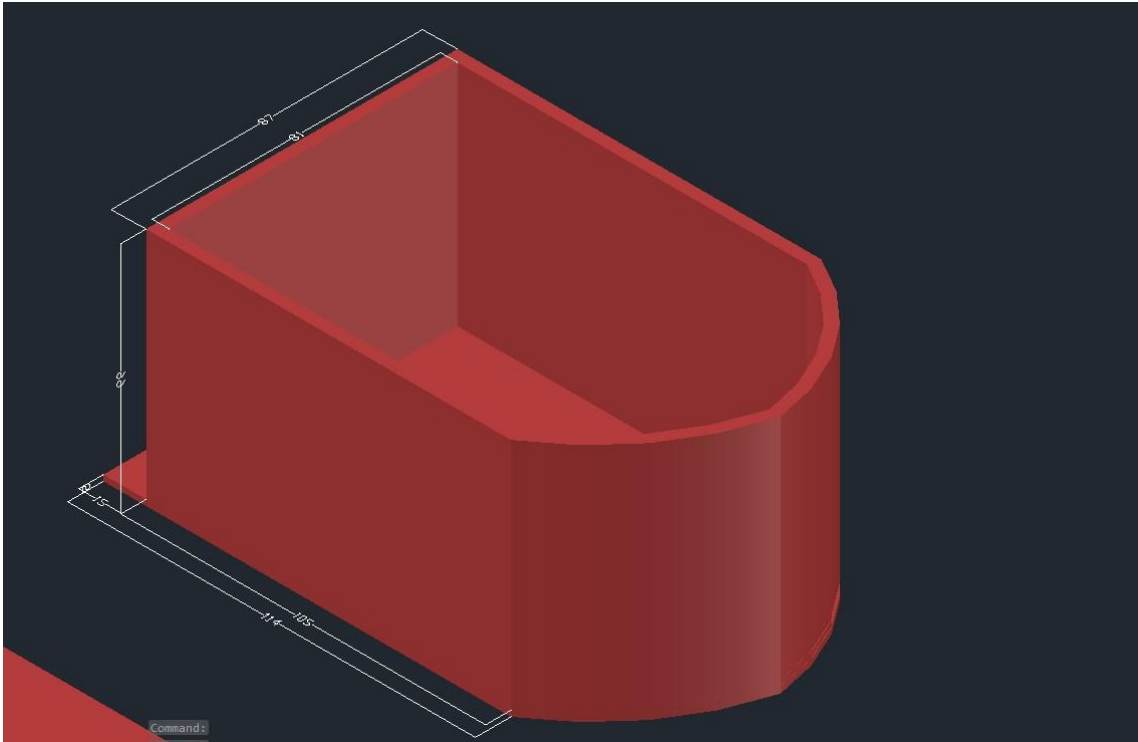


Figure 17 : Process of desining robotic boat by AutoCAD 3

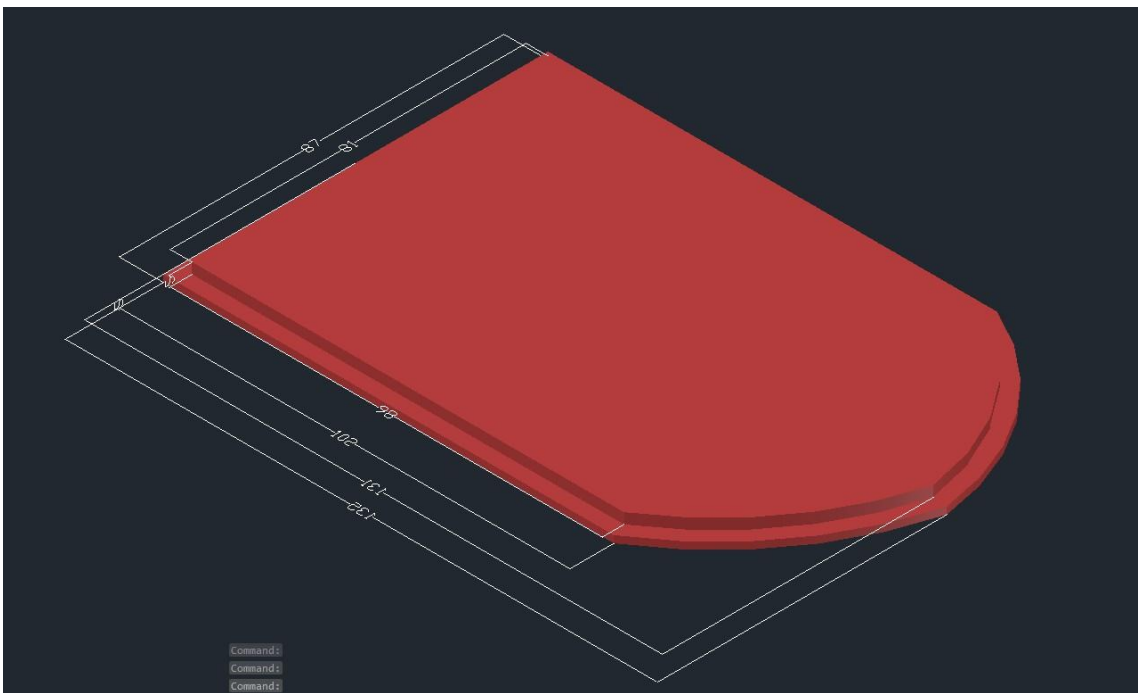


Figure 18 : Process of desining robotic boat by AutoCAD 4

### Protoype robotic boat

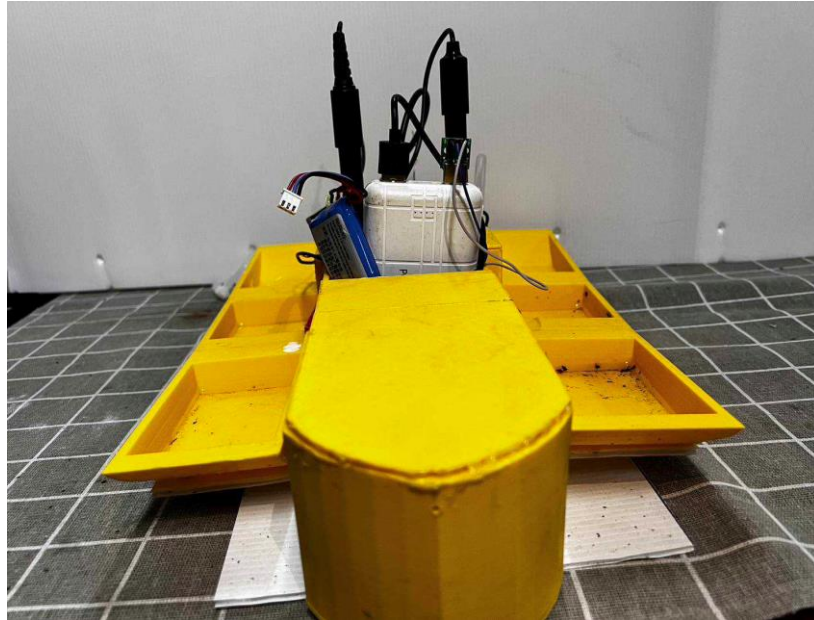


Figure 19 : Front of robot

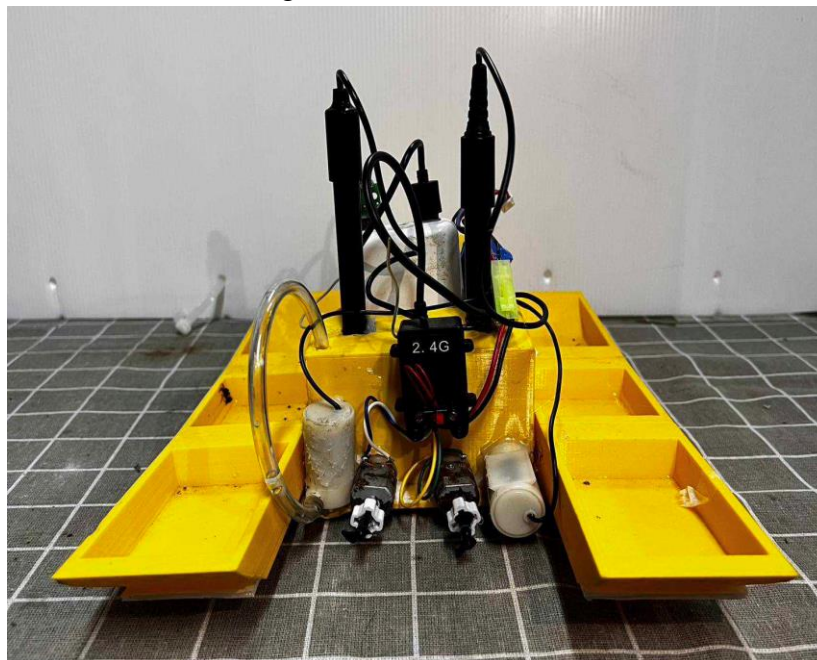


Figure 20: At Behind we have box to collect water and The equipment to inspection water



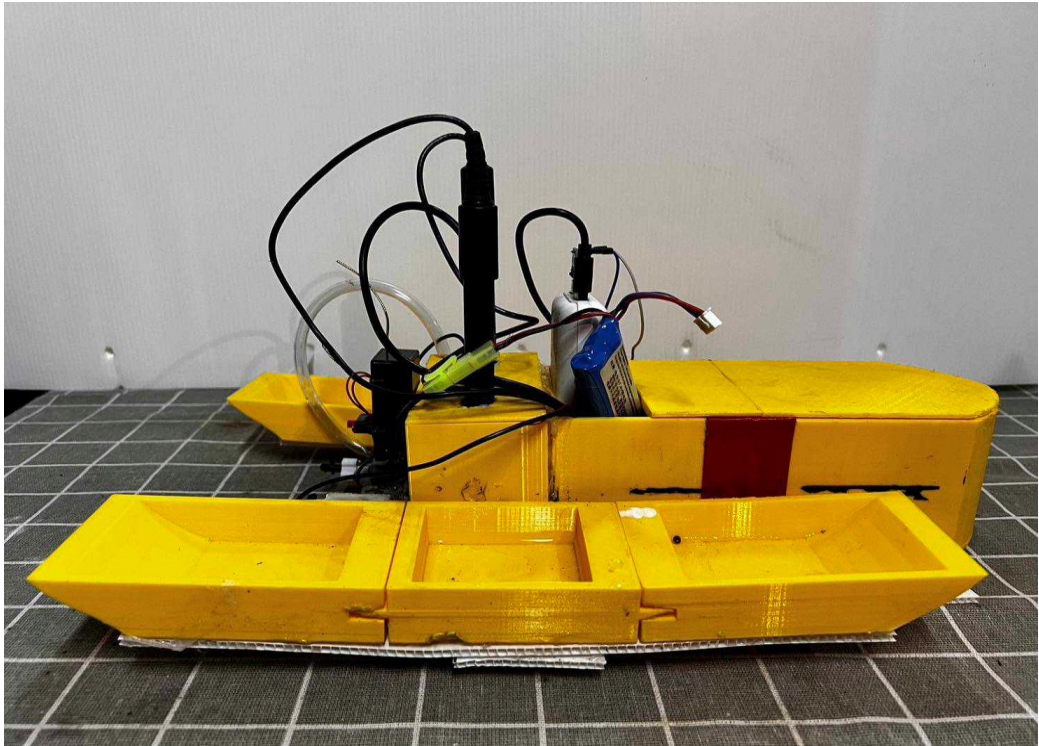


Figure 21 : Side of robot

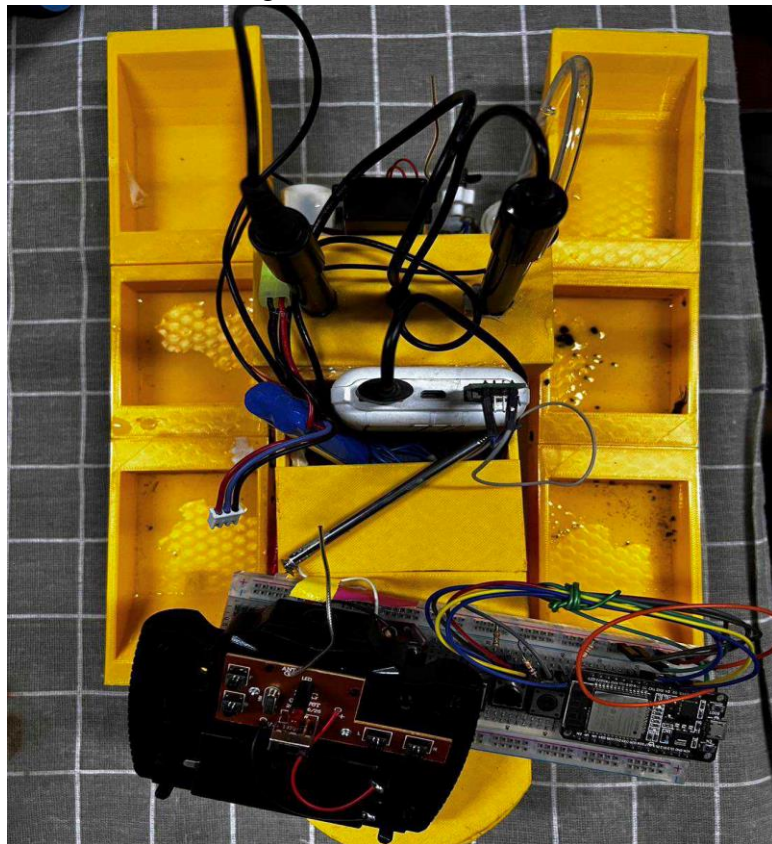


Figure 22 : Component of Robotic boat



Figure 23 : Various components of the robot and remote control

### Testing the performance of the water quality inspection robot

By randomly sampling 5 place within the school premises, using a water quality monitoring robot, and collecting water samples for measurement with a DO meter (Vernier DOV 2-001) and a digital water quality tester Yieryi Professional (pH and TDS), the water quality was compared, yielding results as shown in the graph.

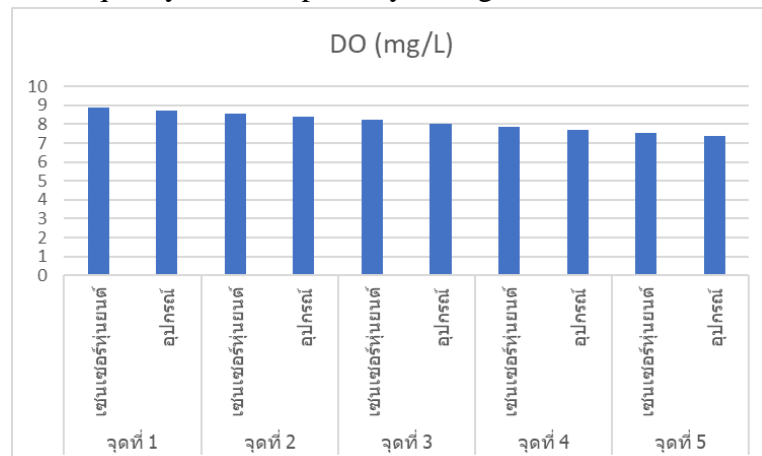


Figure 24 : DO Graph testing results from 5 placr in school

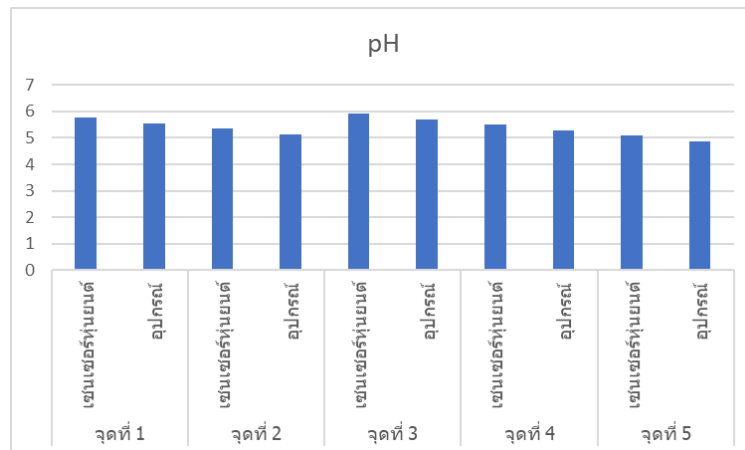


Figure 25 : Ph Graph testing results from 5 placr in school

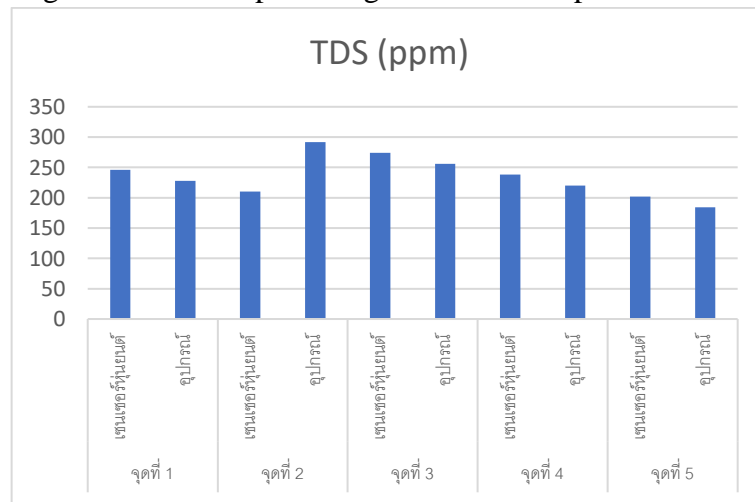


Figure 26 : TDS Graph testing results from 5 place in school

From the graph, when comparing the DO, pH, and TDS values measured using the water quality monitoring robot with those collected samples measured by the DO meter (Vernier DOV 2-001) and the digital water quality tester Yieryi Professional (pH and TDS), it was found that the values were relatively similar. The water quality monitoring robot measured DO values ranging from 7.54 to 8.90 milligrams per liter, pH values ranging from 5.08 to 5.76, and TDS values ranging from 202 to 246 ppm. When measured by the DO meter (Vernier DOV 2-001) and the digital water quality tester Yieryi Professional (pH and TDS), the DO values ranged from 7.37 to 8.73 milligrams per liter, pH values ranged from 4.87 to 5.55, and TDS values ranged from 184 to 228 ppm. These values were consistent across place.

### Result from testing in community water source

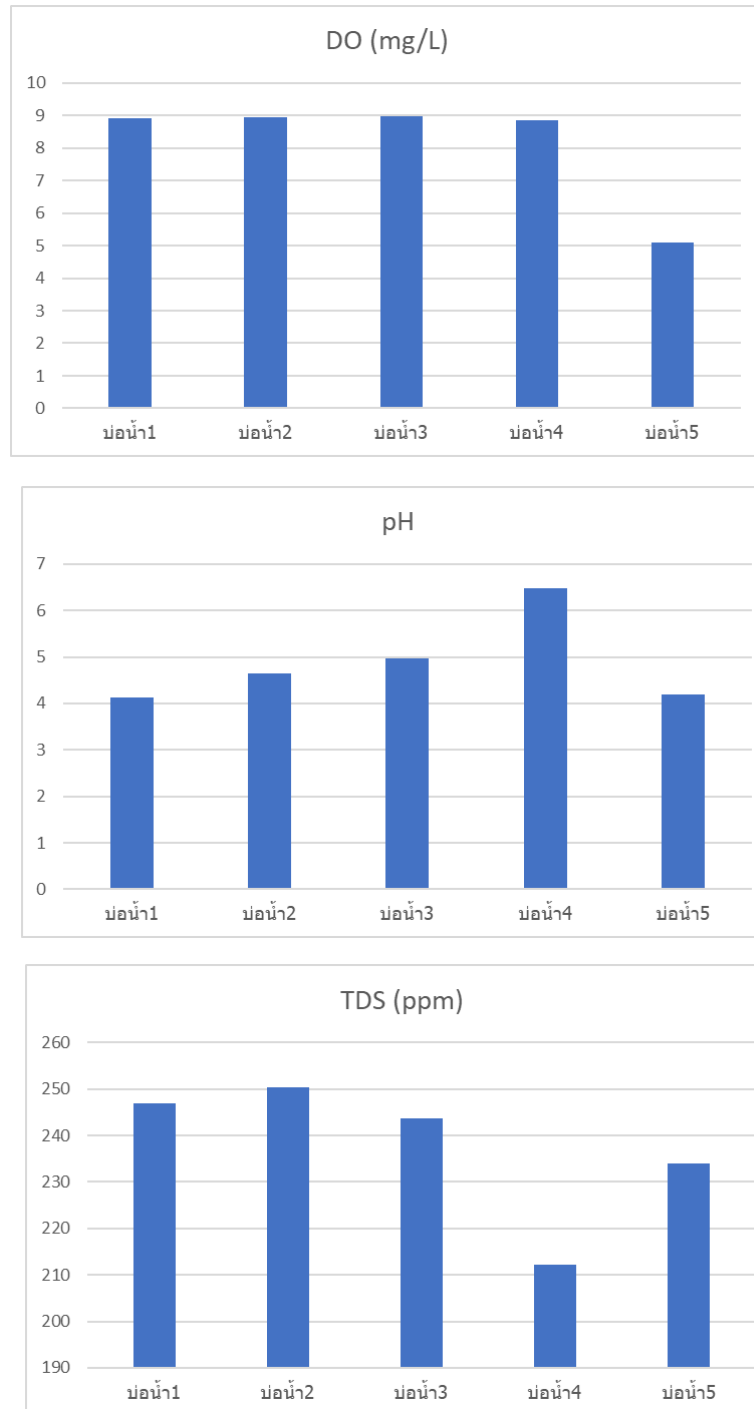


Figure 27 : DO Ph TDS Graph testing results from 5 places in Community water sources

The water quality monitoring robot measured DO values ranging from 5.10 to 8.90 milligrams per liter, pH values ranging from 4.08 to 6.96, and TDS values ranging from 212 to 250 ppm. Measurements were taken approximately 2 meters away from the shore.



**Reference**

**Department of Pollution Control. (2555). Manual for Sample Collection and Soil Sample Analysis for the Prevention and Control of Plant Pests. Bangkok: Department of Pollution Control, Ministry of Natural Resources and Environment.**

**Pramote Maiklad.(2557). Solutions for Thailand's Water Management. [Online]. Available: <https://tdri.or.th/>. [2566, April 10].**

**Department of Irrigation.(2561). huaipung chumrud. [Online]. Available: <http://www.rid.go.th> [2566, April 10].**