



Qatar University

**College of Engineering
Department of Computer Science and
Engineering**

Course Project - CMPE 462 Computer Interfacing

Plant monitoring system

Project Group Member:

Haya Al-Faihani (201801067)

Maha Fekri (201703019)

Nirvana Aladal (201802829)

Table of Contents

List of tables.....	2
List of Figures.....	2
1. Problem Statement and High-Level Architecture.....	4
2. Design Details	4
2.1. Hardware	5
2.1.1. 3D Casing Design	8
2.2. Software	8
3. Implementation and Testing	9
3.1. Actual implementation.....	9
3.2. Complete code	10
3.2.1. Sender code:	10
3.2.2. Receiver:.....	12
3.3. Tests.....	13
4. GUI design.....	14
4.1. GUI image	14
Serial Monitor Output.....	15
4.2. Processing code	15
References	19

List of tables

Table 1: List of components.....	5
Table 2: Test results	13

List of Figures

Figure 1: The high-level architecture of the project.....	4
Figure 2: The connection of the system	7
Figure 3: The schematic of the system	7
Figure 4: Water Tank Base Design	8
Figure 5: Water Tank Lid Design	8
Figure 6: Secondary Development Board	9
Figure 7: Main Development Board	9
Figure 8: Implementation Image	10

Figure 9. Case 1 GUI	14
Figure 10. Case 2 GUI	14
Figure 11. Case 3 GUI	15

1. Problem Statement and High-Level Architecture

Plants are a big part of day-to-day life, from food to being used for aesthetic purposes. Keeping plants thriving needs a lot of time and effort, like knowing when and how much water to provide plants. Today's fast-paced lifestyle causes plant maintenance to be ignored, or in some cases, dealing with big fields with different water absorbance levels in different parts of the field, causing some plants to be watered more than others which affects the crop yield.

This project aims:

- This system can regulate plant water intake based on sensors attached to the plant that can read the soil moisture level and determine whether or not it needs water.
- This system can read the temperature and humidity surrounding the plant and send real-time data to the user.

The system will be built to read sensors, and based on the readings, it will control the water flow with a pump. The readings will be sent to the main board, which acts as a user interface so the user can know the plant parameters. All of the stated functions and connections will be built with the following structure:

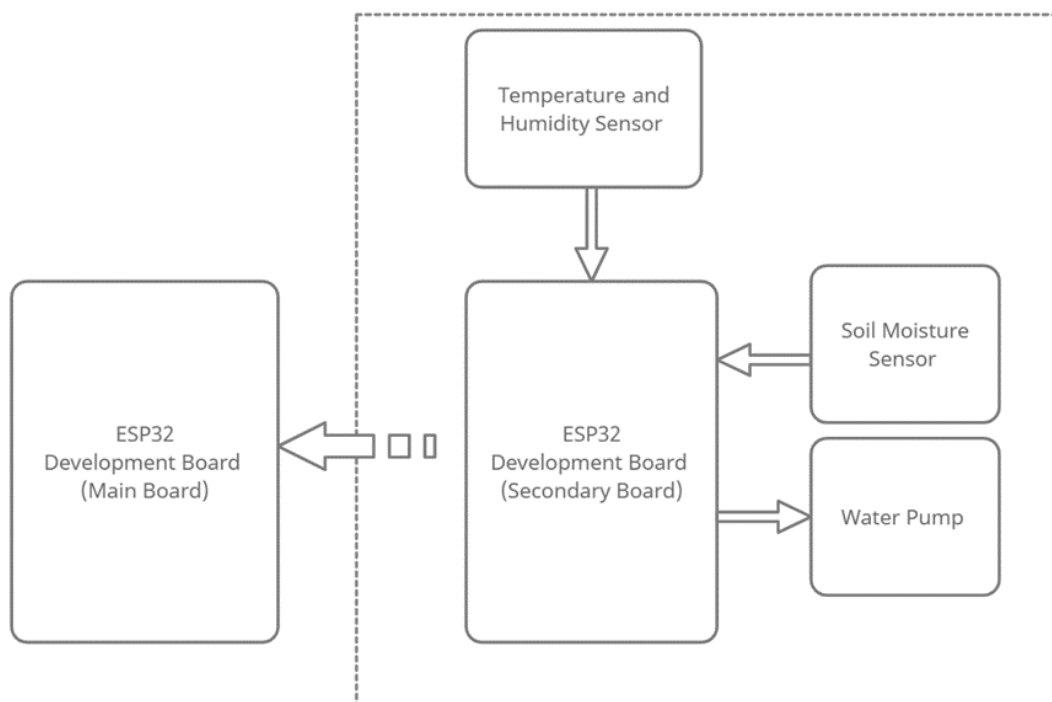


Figure 1: The high-level architecture of the project


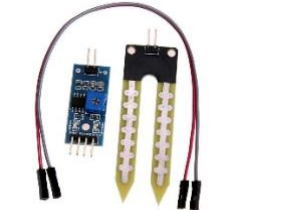
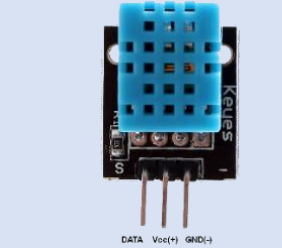
2. Design Details



This part of the report will discuss the speciation of the design reasoning behind it along with the logic that made it work.

2.1. Hardware

This system required using different components to implement the design idea:

Table 1: List of components

Name of Component	Description	Justification	Image
Esp32 Microcontroller [2]	Is a microcontroller used for mobile, wearable electronics, and Internet-of-Things (IoT) applications. Also, has both Bluetooth and Wi-Fi embedded	<ul style="list-style-type: none"> • Scalable to a variety of applications • Meets the system's needs • Small dimensions 	
Soil Moisture sensor [3]	The fork-shaped probe is a conductor that functions as a variable resistor. The resistance changes depending on the amount of water in the soil. As the moisture increase in the soil, the conductivity increases and the resistance decreases.	<ul style="list-style-type: none"> • Availability in the market. • Accurate. 	
DHT11 Temperature and Humidity Sensor [4]	Measures the humidity through the two electrodes when it absorbs water vapor. The ions are released. Therefore, the	<ul style="list-style-type: none"> • Availability in the market. • Accurate. 	

	<p>conductivity between the electrodes increases. When the humidity is high, the resistance between the electrodes is low, and vice versa.</p> <p>The temperature sensor is a thermal resistor. Its resistance varies depending on the temperature.</p>		
Small Submersible Water Pump [5]	<p>It has a motor that is connected to the pump to power an impeller that rotates and pushes water outwards.</p>	<ul style="list-style-type: none"> • Safe usage • relatively small 	
1 Channel Relay Module [6]	<p>Is an interface that allows controlling of high voltages and high currents</p>	<p>Arduino operates at 5V, and it cannot directly control higher voltages therefore using a relay module to control the AC powered devices and Arduino to control the relay</p>	

In terms of microcontrollers, the ESP32 was chosen since it has built-in Bluetooth and WiFi. This is more convenient for communication with a computer. Also, it has a lower cost compared to other microcontrollers. The ESP32 is connected directly to the DHT11 and soil moisture sensors through General-Purpose Input/Output GPIO Because it has built-in Analog to Digital Convertor ADC. In addition, a battery with 9v is used to power up a water pump with an operating range between 7v-12v.

There were several options to establish communication between the two microcontrollers, such as WiFi and Bluetooth. However, the ESP-NOW was chosen as a connectionless

communication protocol. It enables one-way communication without using WiFi. Therefore, it provides faster data transmission because it does not need to connect to a Wi-Fi Access Point.

The connection of the components was illustrated with the diagram that shows the overall connectivity of the hardware implementation of the project. This was demonstrated using the datasheets for each sensor to establish the connection between the sensors and the microcontroller, which resulted in the following diagrams:

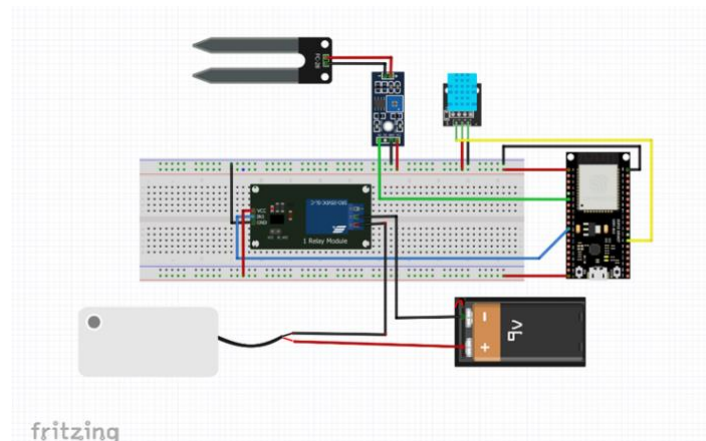


Figure 2: The connection of the system

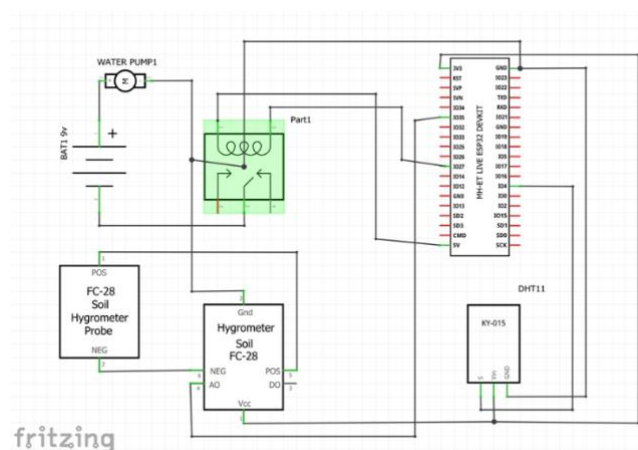


Figure 3: The schematic of the system

2.1.1. 3D Casing Design

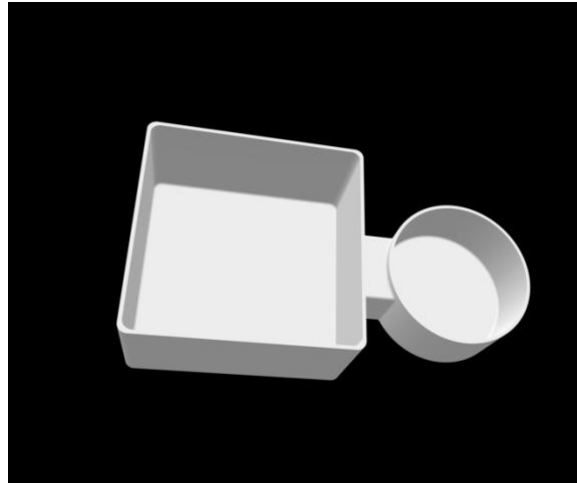


Figure 4: Water Tank Base Design

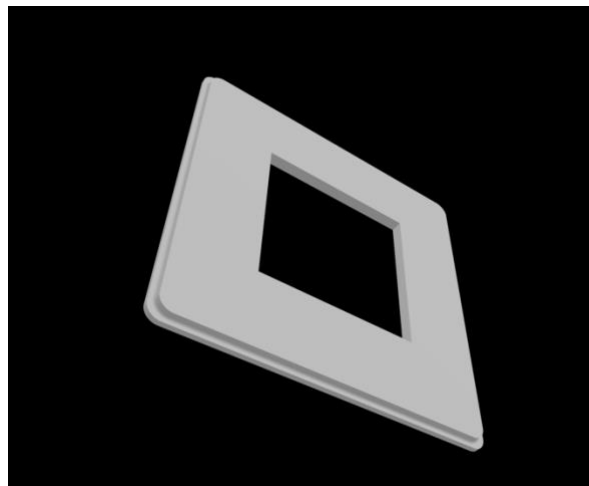


Figure 5: Water Tank Lid Design

2.2. Software

The system uses the Arduino IDE to program the microcontroller, along with the Processing IDE, which will be used to develop a user interface because it is compatible.

The program works by using the soil moisture sensor in this system to detect the moisture level in the soil. If the soil becomes dry, the sensor detects a low moisture level and instantly activates the water pump to deliver water to the plant. When the plant receives enough water and the soil becomes wet, the sensor detects sufficient moisture. The water pump will thereafter be turned off automatically; the operation can be represented by the following flowchart:

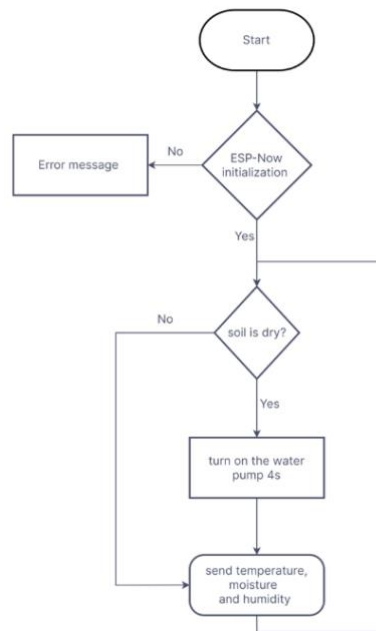


Figure 6: Secondary Development Board

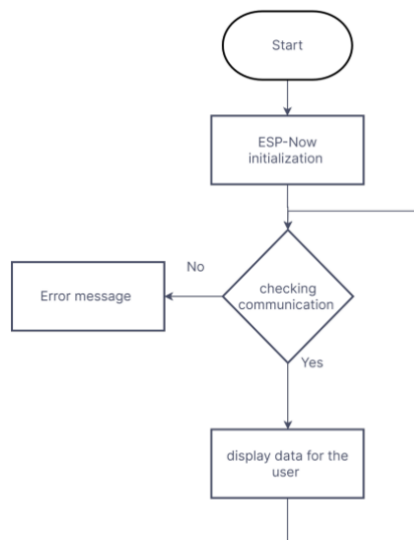


Figure 7: Main Development Board

3. Implementation and Testing

This part of the report documents the actual system from coding, system connection, to testing.

3.1. Actual implementation

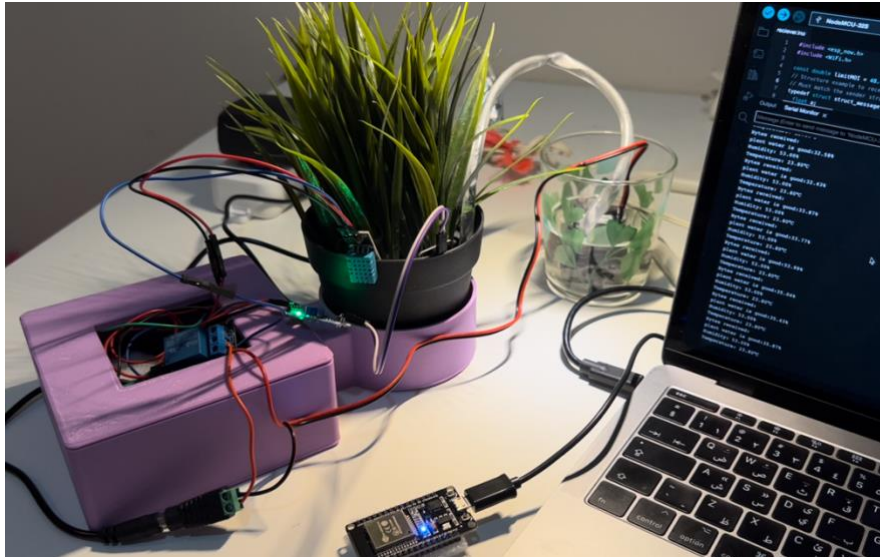


Figure 8: Implementation Image

3.2. Complete code

The code composes of two different parts the sender (secondary), and the receiver (main).

3.2.1. Sender code:

```
//establishing pins
#define MOI_PIN 35
#define RELAY_PIN 27
#define DHTPIN 4
//adding libraries
#include <esp_now.h>
#include <WiFi.h>
#include "DHT.h"
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
//RECEIVER MAC Address
uint8_t broadcastAddress[] = {0x94, 0xB5, 0x55, 0x2C, 0x53, 0x24};
//message structer
typedef struct struct_message {
    float a;
    float b;
    float c;
} struct_message;
struct_message myData;
esp_now_peer_info_t peerInfo;
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");
}
//soil moister level
const double limitMOI = 48.0;
void setup() {
```

```

Serial.begin(115200);
pinMode(MOI_PIN, INPUT);
dht.begin();
pinMode(RELAY_PIN, OUTPUT);

WiFi.mode(WIFI_STA);
if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
}
esp_now_register_send_cb(OnDataSent);
memcpy(peerInfo.peer_addr, broadcastAddress, 6);
peerInfo.channel = 0;
peerInfo.encrypt = false;
if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
}
}

void loop() {
    float valueMOI;
    //reading soil moister
    valueMOI = (analogRead(MOI_PIN) / 4095.0) * 100;
    if (valueMOI > limitMOI)
    {
        digitalWrite(RELAY_PIN, HIGH); // turn on pump 4 seconds
        delay(4000);
        digitalWrite(RELAY_PIN, LOW);
    }
    //reading Humidity
    float h = dht.readHumidity();
    // Read temperature as Celsius (the default)
    float t = dht.readTemperature();
    //data ready to send
    myData.a = valueMOI;
    myData.b = h;
    myData.c = t;
    //sending data
    esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &myData,
    sizeof(myData));

    if (result == ESP_OK) {
        Serial.println("Sent with success");
    }
    else {
        Serial.println("Error sending the data");
    }

    delay(2000);
}

```

3.2.2. Receiver:

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

const double limitMOI = 48.0, limitWAT = 100.0;
// Structure example to receive data
// Must match the sender structure
typedef struct struct_message {
    float a;
    float b;
    float c;
} struct_message;

// Create a struct_message called myData
struct_message myData;

// callback function that will be executed when data is received
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
    memcpy(&myData, incomingData, sizeof(myData));
    //Serial.println("Bytes received: ");
    //soil moi
    if (myData.a < limitMOI)
    {
        //Serial.print("plant water is good:");
        Serial.print(myData.a);
        Serial.print(",");
    }
    else
    {
        //Serial.print("plant water is bad:");
        Serial.print(myData.a);
        Serial.print(",");
    }
    //tem&hum
    //Serial.print("Humidity: ");
    Serial.print(myData.b);
    Serial.print(",");
    //Serial.print("Temperature: ");
    Serial.print(myData.c);
    Serial.print(",");
}

void setup() {
    // Initialize Serial Monitor
    Serial.begin(115200);

    // Set device as a Wi-Fi Station
    WiFi.mode(WIFI_STA);

    // Init ESP-NOW
    if (esp_now_init() != ESP_OK) {
```

```

    Serial.println("Error initializing ESP-NOW");
    return;
}

// Once ESPNow is successfully Init, we will register for recv CB to
// get recv packer info
esp_now_register_recv_cb(OnDataRecv);
}
void loop() {
}

```

3.3. Tests

The test of the system was divided into two parts. The first one is the ability of the system to read the parameters and react to the changes fittingly, and the second part is the connection between the two boards.

Table 2: Test results

Test description	Test feedback	Results
The test was conducted over a week with two plants where the system was left to take care of one plant watering, and the second one was taken care of by the owner.	The system could detect when to water the plant and how much water was needed, and the plant's needs were met. On the other hand, the second plant was watered less frequently, using more water each time, overall consuming more water from the water tank than the first plant, along with being neglected on day 4 of the testing as the user was not home.	Positive.
The two esp32 boards were placed in two different rooms, and their connection was tested over 20-minutes.	The connection initialization took a few seconds to create a data path between them, and after that, the real-time data was sent with an average of 2-second delay.	Positive.

The system behaves as expected; it can detect the plant's parameters and react appropriately to the changes, along with a strong intercommunication. Some improvements can be made, which are:

1. Having a many-to-one connection.
2. Adding more features to help the plant, for example, a system to control the light intake.
3. Better compact design.

4. GUI design

4.1. GUI image

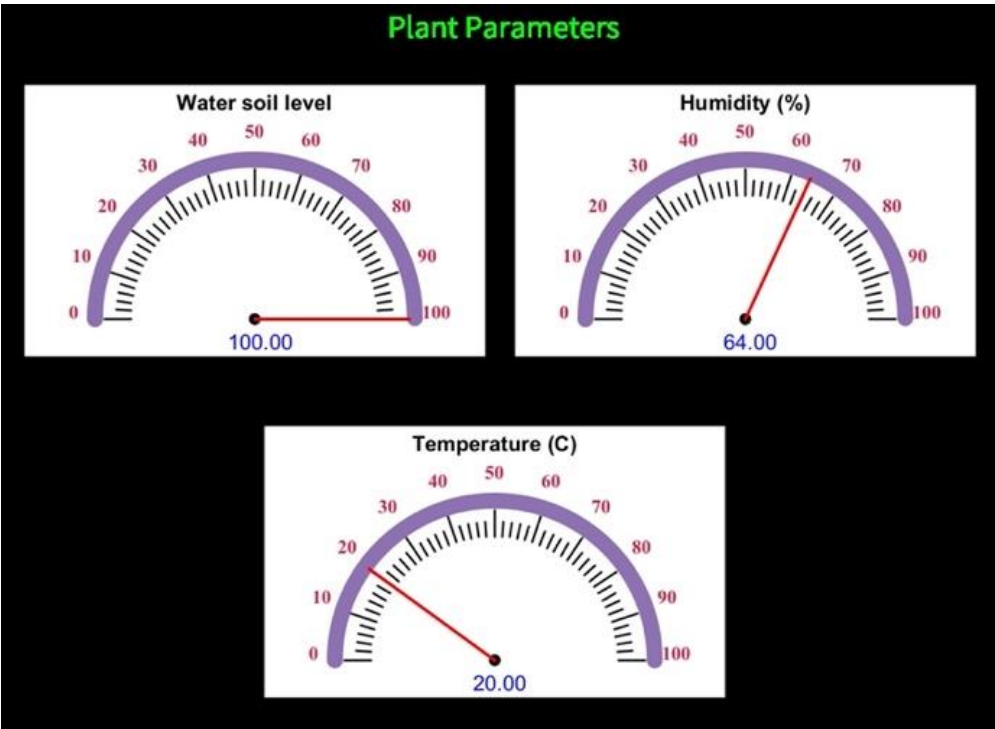


Figure 9. Case 1 GUI

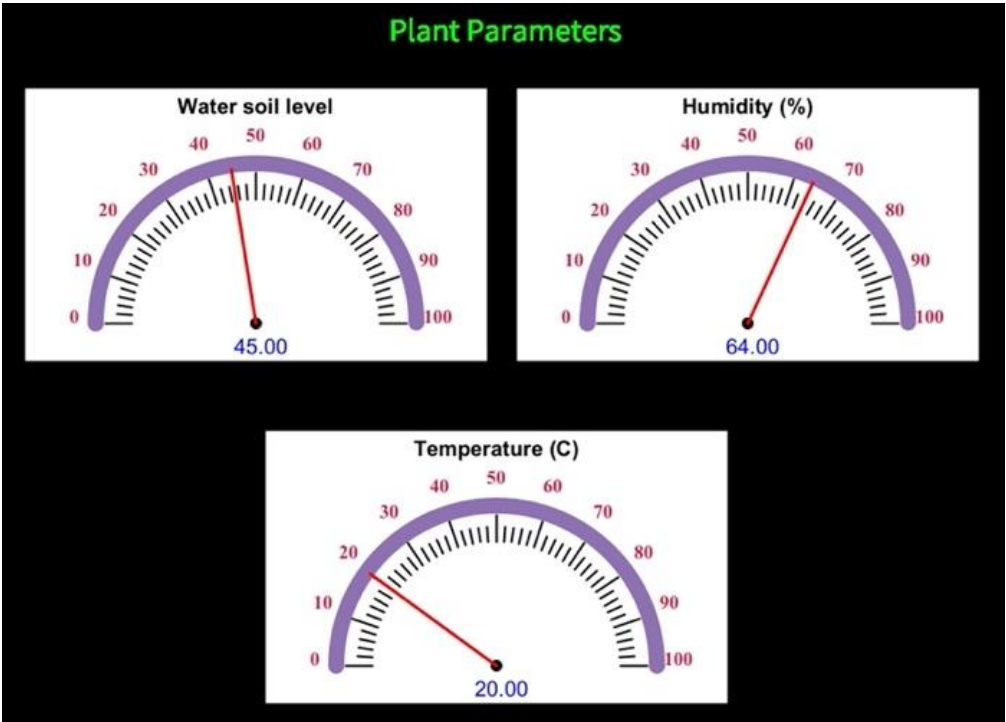


Figure 10. Case 2 GUI

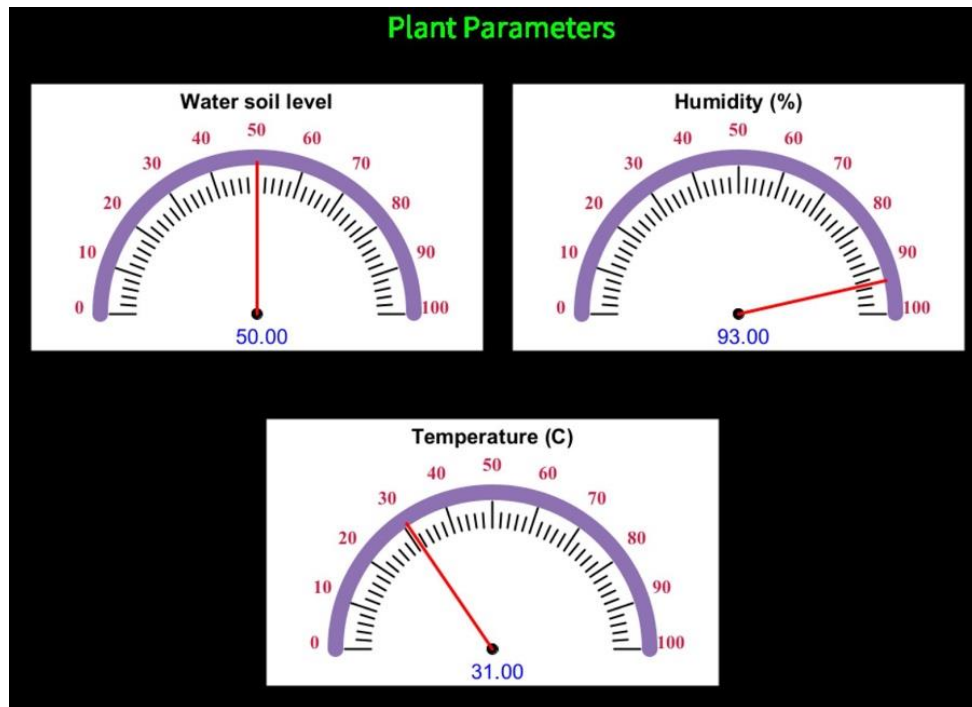


Figure 11. Case 3 GUI

Serial Monitor Output

Water soil level: 100.0 Humidity: 64.0% Temperature: 20.0 C
 Water soil level: 100.0 Humidity: 64.0% Temperature: 20.0 C
 Water soil level: 100.0 Humidity: 64.0% Temperature: 20.0 C
 Water soil level: 100.0 Humidity: 64.0% Temperature: 20.0 C
 Water soil level: 100.0 Humidity: 64.0% Temperature: 20.0 C
 Water soil level: 100.0 Humidity: 64.0% Temperature: 20.0 C
 Water soil level: 35.24 Humidity: 64.0% Temperature: 20.0 C
 Water soil level: 38.27 Humidity: 64.0% Temperature: 20.0 C
 Water soil level: 42.95 Humidity: 64.0% Temperature: 20.0 C
 Water soil level: 45.54 Humidity: 64.0% Temperature: 20.0 C
 Water soil level: 47.72 Humidity: 64.0% Temperature: 20.0 C
 Water soil level: 48.99 Humidity: 64.0% Temperature: 20.0 C
 Water soil level: 50.5 Humidity: 64.0% Temperature: 20.0 C
 Water soil level: 51.18 Humidity: 64.0% Temperature: 20.0 C
 Water soil level: 49.62 Humidity: 64.0% Temperature: 20.0 C
 Water soil level: 51.55 Humidity: 92.0% Temperature: 31.0 C
 Water soil level: 51.33 Humidity: 93.0% Temperature: 34.0 C
 Water soil level: 51.14 Humidity: 94.0% Temperature: 32.0 C
 Water soil level: 50.53 Humidity: 93.0% Temperature: 31.0 C
 Water soil level: 50.74 Humidity: 93.0% Temperature: 30.0 C
 Water soil level: 49.79 Humidity: 93.0% Temperature: 29.0 C

4.2. Processing code

```
// Lets create a simple meter
```

```
// Import Meter library
import meter.*;
```

```

// Import serial library
import processing.serial.*;

Serial port; // Define a port

Meter m, m2, m3;

void setup(){
  // First we need to create a empty window
  size(950, 700); // Size of the window (width, height)
  background(0, 0, 0); // Background color of window (R,G,B)

  // Create new port
  port = new Serial(this, "COM3", 115200); //name of the port would be different for linux

  // Lets add a default meter to empty window
  // TEMPERATURE METER
  m = new Meter(this, 25, 80); // here 25, 10 are x and y coordinates of meter's upper left
  corner

  m.setTitleFontSize(20);
  m.setTitleFontName("Arial bold");
  m.setTitle("Water soil level");

  // Change meter scale values
  String[] scaleLabels = {"0", "10", "20", "30", "40", "50", "60", "70", "80", "90", "100"};
  m.setScaleLabels(scaleLabels);
  m.setScaleFontSize(18);
  m.setScaleFontName("Times new roman bold");
  m.setScaleFontColor(color(200, 30, 70));

  // We can also display the value of meter
  m.setDisplayDigitalMeterValue(true);

  // Lets do some more modifications so our meter looks nice
  m.setArcColor(color(141, 113, 178));
  m.setArcThickness(15);

  m.setMaxScaleValue(100);

  m.setMinInputSignal(0);
  m.setMaxInputSignal(100);

  m.setNeedleThickness(3);

  // HUMIDITY METER
  // lets take some refference from first meter
  int mx = m.getMeterX(); // x coordinate of m
  int my = m.getMeterY(); // y coordinate of m

```



```

int mw = m.getMeterWidth();

m2 = new Meter(this, mx + mw + 20, my);

m2.setTitleFontSize(20);
m2.setTitleFontName("Arial bold");
m2.setTitle("Humidity (%)");

// Change meter scale values
String[] scaleLabels2 = {"0", "10", "20", "30", "40", "50", "60", "70", "80", "90", "100"};
m2.setScaleLabels(scaleLabels2);
m2.setScaleFontSize(18);
m2.setScaleFontName("Times new roman bold");
m2.setScaleFontColor(color(200, 30, 70));

// We can also display the value of meter
m2.setDisplayDigitalMeterValue(true);

// Lets do some more modifications so our meter looks nice
m2.setArcColor(color(141, 113, 178));
m2.setArcThickness(15);

m2.setMaxScaleValue(100);

m2.setMinInputSignal(0);
m2.setMaxInputSignal(100);

m2.setNeedleThickness(3);

////Potentiometer

// lets take some refference from first meter
int mx1 = m.getMeterX(); // x coordinate of m
int my1 = m.getMeterY(); // y coordinate of m
int mw1 = m.getMeterWidth();

m3 = new Meter(this, 250, 400);

m3.setTitleFontSize(20);
m3.setTitleFontName("Arial bold");
m3.setTitle("Temperature (C)");

// Change meter scale values
String[] scaleLabels3 = {"0", "10", "20", "30", "40", "50", "60", "70", "80", "90", "100"};
m3.setScaleLabels(scaleLabels3);
m3.setScaleFontSize(18);
m3.setScaleFontName("Times new roman bold");
m3.setScaleFontColor(color(200, 30, 70));

// We can also display the value of meter

```

```

m3.setDisplayDigitalMeterValue(true);

// Lets do some more modifications so our meter looks nice
m3.setArcColor(color(141, 113, 178));
m3.setArcThickness(15);

m3.setMaxScaleValue(100);

m3.setMinInputSignal(0);
m3.setMaxInputSignal(100);

m3.setNeedleThickness(3);
}

void draw(){
  // Lets give title to our window
  textSize(30);
  fill(0, 255, 0); // Font color , (r,g,b)
  text("Plant Parameters", 370, 40); // ("text", x, y)

  if (port.available() > 0){
    String val = port.readString(); // read incoming string on serial port
    // First we need to separate temperature and humidity values
    String[] list = split(val, ','); // splits value separated by ','
    float temp = float(list[0]); // first value is Temperature
    float hum = float(list[1]); // second value is Humidity
    float pot = int(list[2]); // third value potentiometer
    m.updateMeter(int(temp)); // int is used due to updateMeter accepts only int values
    m2.updateMeter(int(hum));
    m3.updateMeter(int(pot));
    println("Water soil level: " + temp + " Humidity: " + hum+ "%"+ " Temperature: "+pot+ "
C ");
  }
}

```

References

- [1] P Vaishnavi, Y Lakshmi Sai Charitha, T Jayasri, Manchikanti Bhumika, and Nelakurthi Sudheer Kumar, "Automatic Watering System using Soil Moisture Sensor and RTC timer with Arduino," *International Journal of Advanced Research in Science, Communication and Technology*, pp. 55–69, Feb. 2022, doi: 10.48175/ijarsct-2507.
- [2] "ESP32 Development Board," *Voltaat*. <https://www.voltaat.com/products/esp32-development-board> (accessed Nov. 16, 2022).
- [3] "Soil Moisture Sensor," *Voltaat*. <https://www.voltaat.com/products/soil-moisture-sensor> (accessed Nov. 16, 2022).
- [4] "DHT11 Temperature & Humidity Sensor," *Voltaat*. <https://www.voltaat.com/products/dht11-temperature-humidity-sensor> (accessed Nov. 16, 2022).
- [5] "Small Submersible Water Pump," *Voltaat*. <https://www.voltaat.com/products/small-submersible-water-pump> (accessed Nov. 16, 2022).
- [6] "1 Channel Relay Module," *Voltaat*. <https://www.voltaat.com/products/1-channel-relay-module> (accessed Nov. 16, 2022).