Proceedings of the 12th International Conference of Control Systems, and Robotics (CDSR 2025)
July 13, 2025 - July 15, 2025 | Imperial College London Conference Center, London, United Kingdom
Paper No. 301
DOI: 10.11159/cdsr25.301

An Efficient Internet of Things (IoT) Based Agriculture Automation System

Harrison Carranza[1], Aparicio Carranza[2], Miguel Bustamante[2]
[1]Bronx Community College of the The City University of New York
2155 University Avenue, Bronx, NY, USA
Harrison.Carranza@bcc.cuny.edu
[2]Vaughn College of Aeronautics and Technology
86-01 23rd Avenue, Flushing, NY, USA
Aparicio.Carranza@vaughn.edu, Miguel.Bustamante@vaughn.edu

Abstract **-** Due to overwork farmers misallocate their time and this leads to the accidental neglect of plants, and they die due to over or under watering, etc., resulting in lots of food being wasted. We have designed and implemented an efficient Internet of Things (IoT) Agricultural System using the power of ESP32 module, temperature and humidity sensors, solenoid valves, and a convenient User Interface (UI). The communication subsystem module uses the Wi-Fi Networking for compatibility with more devices. The web interface has been written in C#, a low-level web development language, to use lower system resources to reduce the overall cost.

Keywords**:** ESP32, IoT, WiFi, C#

1. Introduction

To achieve the efficiency of our IoT based agriculture system, we have developed and implemented a simple shared protocol with an ESP32 module and a Server that can talk to each other, a web interface where the user can configure and monitor the system, a database in which the user can look at the history and use it for planning [1]. Due to not having an actual farm readily available, we have performed the testing on a small houseplant instead. The shared protocol has been our main focus here because without it the system will not be able to work together, it must be able to run on small devices that it is based on WiFi and TCP connections (*small packets containing information on sensors and actions over Transmission Control Protocol*). The backend in the server allows another developer to add a more complex and user-friendly interface to the system [2].

Our role in this operation has been to deploy the microcontroller ESP32 as shown in Fig. 1 to feed power to DHT11, Photosensitive Diode, water level sensors and control pump. The ESP32 is powered by the software, Arduino IDE and the ESP32 microcontroller has the power to create a web server [3]. DHT11 reads the temperature of the plant and produces a reading. The Photosensitive Diode sensor detects light ray sensitivity. It allows the user to know when the plant requires some light source depending on the sensitivity of the ray. The water level detects the amount of water in the soil. Lastly, and not the least the control pump enables the water to flow from the tube to the plant's soil. If the water temperature and light readings reaches above the maximum capacity within normal standards according to the plant's need, then the website lets the ESP32 to immediately stop the source. However, any reading that produces a value within a range where the plant needs these resources, the ESP32 enables light and water source.

The web interface has been written using C#, this interface represents a contract between an object and its user. Interfaces in C# help us achieve our goals allowing objects to interact with each other [4]. The C# interface acts as a blueprint for our methods, and events and makes it more flexible the realization of our project. Additionally, C# interface can improve extensibility where other developers might extend the program. Finally, C# interfaces are ideal to create and implement our solution, since they can be inherited from other interfaces [5].

The creation of algorithms is a fundamental task for managing and interacting with IoT Devices. It provides a powerful means to store, retrieve, manipulate, and control data within these systems. Algorithms are ubiquitous in the world of data management, used in a wide range of applications and industries to work with structured data efficiently. Whether you are a database administrator, data analyst, and software developer, or business professional, understanding algorithms are essential

for effectively harnessing the power of IoT systems and the data collected from them makes us informed decisions based on the collected data.

In Section 1, we presented the main objectives for the realization of our project, next in Section 2, we discuss about the Hardware and Tools Employed, in Section 3, The Wireless Connectivity is described, in Section 4, The Sensors and Motors are presented, in Section 5, Algorithmic Methodologies is touched, then in Section 6, Web Interface Implementation is described, and finally our Conclusion is included in Section 7.

2. Hardware and Tools Employed

In Fig. 1, the following components are shown: on the left side of the panel, the ESP32-C6-EVB, a Capacitive soil moisture sensor, DHT11, motors, Photosensitive Diode sensor is on the middle of the panel and lastly all the hardware to work together on the last panel.

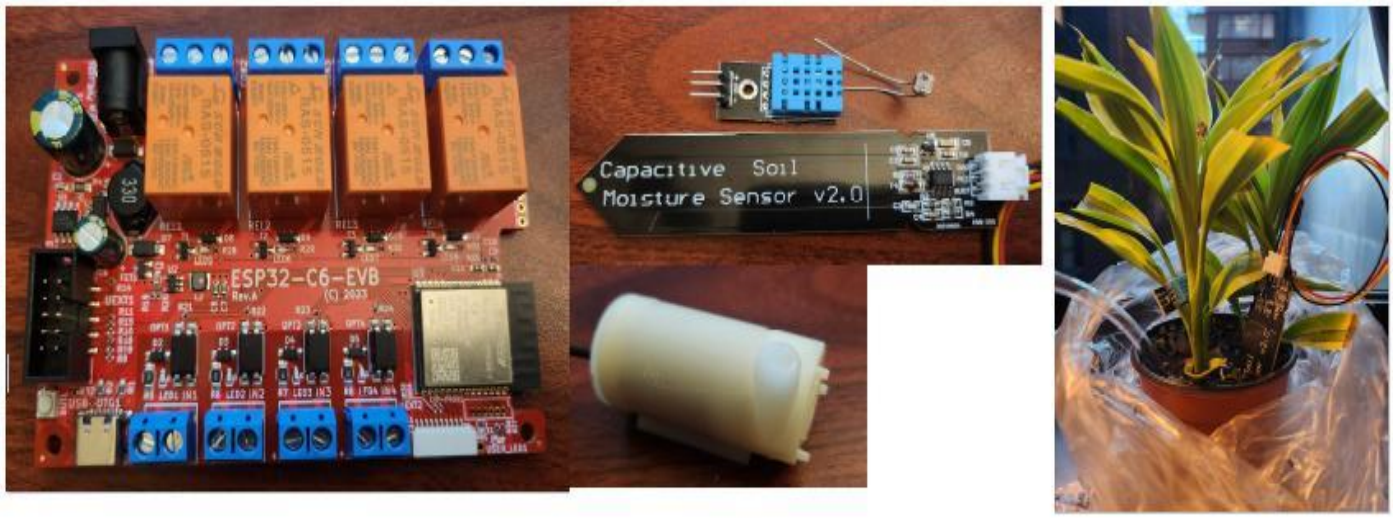

Fig. 1 Picture of our hardware components

The ESP32-C6-EVB has 4 built in relays from olimex, the module with the blue plastic is a temperature and humidity sensor. The arrow shaped board under it is a moisture sensor that can measure soil moisture, next to the blue module is a small light sensor (*photoresistor*) and a pump from Amazon. On the right is the subject of the experiment, a plant from the farmer's market. We also added an ESP32-S3, which is seen later in the paper when the full system is completed.

3. The Wireless Connectivity

For the communication subsystem, we implemented a simple shared protocol for the ESP32 and a Server to converse with each other. A web interface was also implemented for the user to configure and monitor the system, a database for planning and look at the history and its usage. Due to not having an actual farm, the testing was performed on a small houseplant instead.

The shared protocol is one of our main component and focus here because without this the system will not be able to work in conjunction with the rest of the systems. It is meant to run on small devices and is based on WiFi and TCP connections. The backend of the server allows other developers to add a more complex and user friendly interfaces to the system. We successfully made our WiFi to work by modifying an example found in GitHub and replaced the code with one that sends sensor data to a server, as well as load settings and rules [2].

To simplify the wireless system, we used host names rather than IP addresses, this approach provides security advantages too as it makes harder to access the IP system of the user's network from one of the edge IoT devices. We used a dotnet C#

backend because low-level codes can be written into TCP servers and web servers using the same language with less effort. A segment of the code is shown in Fig. 2.

```
// This is the code for the CET4960 Group Project for the ESP32 - Carranza
// https://projecthub.arduino.cc/arcaegecengiz/using-dht11-12f621
// Used this for DHT sensor
// SSID & Password
const char* ssid = "*enterssidhere*";  // Enter your SSID here
const char* password = "*enterpasshere*";  //Enter your Password here
const char * host = "192.168.1.75"; // server to connect to, the ip of the computer running visualstudio
String devicename = "deskplant controller";
int wateringrate = 4000; // How many ms the pump should be on for, pumps fast so not for long.

#include <dht11.h>
#include <WiFi.h>
#include <WiFiMulti.h>

#define DHT11PIN 6
dht11 DHT11;

WiFiMulti WiFiMulti;

#define soilsensor 7
#define lightsensor 5
#define relaypin 14

void setup()
{
  pinMode(soilsensor, INPUT);
  pinMode(lightsensor, INPUT);
  pinMode(relaypin, OUTPUT);
  digitalWrite(relaypin, LOW);
    delay(10);

    // We start by connecting to a WiFi network
    WiFiMulti.addAP(ssid, password);


    while(WiFiMulti.run() != WL_CONNECTED) {
        delay(500);
    }

    delay(500);
}


void loop() {

  int chk = DHT11.read(DHT11PIN);

  // Submission Example: name.MM.DD.YYYY.HR.MIN.SEC.soil.temp.light.humidity.motoron.lighton

  int humidity = DHT11.humidity;
  int temp = (DHT11.temperature*1.8) + 32;
  int soil = analogRead(soilsensor) / 41;
  int light = 100 - (analogRead(lightsensor) / 41);
```

Fig. 2 Arduino code for the sensor board using the ESP32-S3.

4. The Sensors and Motors

The purpose of the deployed microcontroller ESP32, shown in Fig. 1, has been to feed power to sensor modules, those are DHT11, Photosensitive Diode, water level sensors and control pump. DHT11 sensor module is the component that reads the humidity and temperature of the object. According to the DHT 11 datasheet, it requires to be powered by five Volts power supply and established a GND connection to run. Once these connections are established and instructions are sent to the module through the microcontroller, it transmits temperature data in the range of Zero to Fifty+ degrees Celsius and humidity range from twenty to ninety percent. The DHT then reads the plant's temperature and humidity.

Next, we move on to the Photosensitive Diode sensor. According to the data-sheet of this module, the Photosensitive Diode sensor function is to detect light ray sensitivity. This sensor requires at least a 5-Volts power supply and GND connection to function properly. It has features that contain high sensitivity on the light detection and daylight blocking filters with the ranges of 850 *nm* to 950 *nm* emitters. It also permits the user to know when the plant requires some light source depending on the sensitivity of the ray.

The Capacitive Soil moisture sensor function is described next, which detects the amount of water in the soil. Lastly, the control pump enables the water to flow from the tube to the plant's soil. According to this module, it requires at least 3.3-Volt power supply and GND connection to be established to function properly. The reading values are the amount of water soiled on the plant, the dryness of the soil and the amount of water in the range of zero percent RH to hundred percent RH.

We implemented our system on two breadboards where the following system/components were integrated: ESP32, DHT11 sensor, motor, and photosensitive diode sensor. All the servers and motor share the same Ground connection with the microcontroller. The Photosensitive Diode output pin establishes a connection to GPIO5 pin of the ESP32

microcontroller. The DHT11 output pin establishes as a connection to GPIO6 pin of the ESP32. The Water level sensor output pin makes a connection to GPIO7 pin of the ESP32. The motor's output pin makes a connection to GPIO16 pin of the ESP32, later made into an indicator LED with a separate board just for relay due to electrical problems. If the water, temperature, light readings reached above the max capacity within normal standard according to the plant's need then the website lets the ESP32 immediately to stop the source. However, any readings producing a value within a range where the plant needs these resources, the ESP32 enables light and water source. The assembled system is shown in Fig. 3.

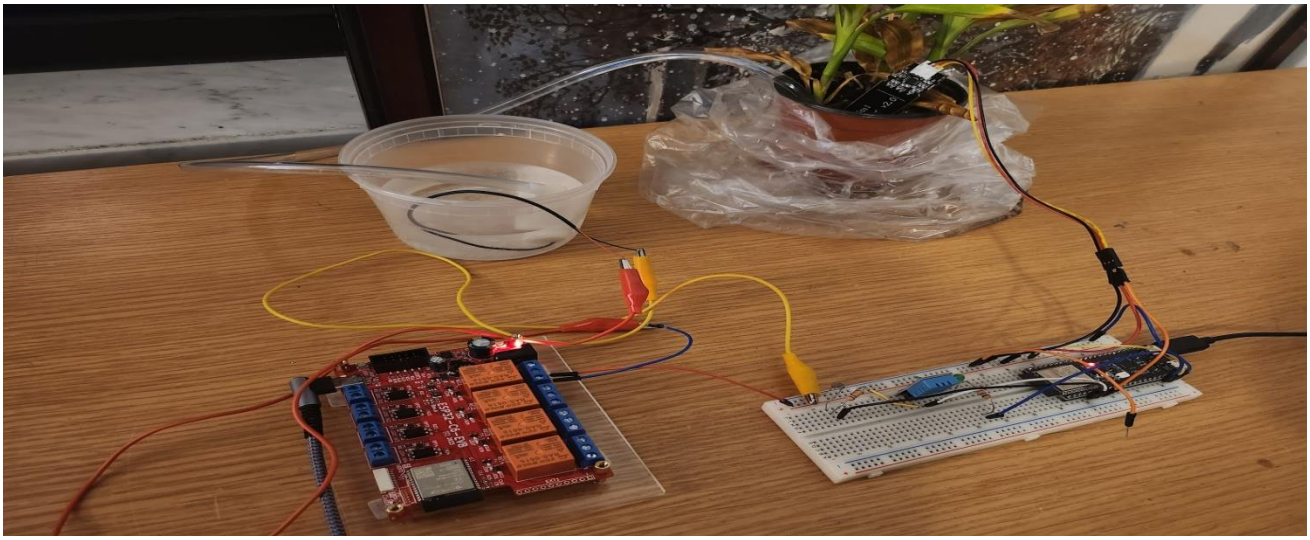

Fig. 3 The Complete System Assembled

5. Algorithmic Methodologies Employed

Next, we present the functional methodologies employed during our implementation that were useful to combine all the data from these systems. This information is shown in Fig. 4.

```
        int i;

        // Loop to receive all the data sent by the client.
        while ((i = stream.Read(bytes, 0, bytes.Length)) != 0)
        {
            // Translate data bytes to a ASCII string.
            data = System.Text.Encoding.ASCII.GetString(bytes, 0, i);
            Console.WriteLine("Received: {0}", data);
            // Process the data sent by the client.
            string dotsep_submission = data;
            string[] submission = dotsep_submission.Split(new string[] { "." }, StringSplitOptions.None);
            // Submission Example: name.MM.DD.YYYY.HR.MIN.SEC.soil.temp.light.humidity.motoron.lighton
            Globals.iotdevice_name = submission[0];
            Globals.Timestamp = new DateTime(int.Parse(submission[3]), int.Parse(submission[1]), int.Parse(submission[2]), int.Pa
            Globals.soil = int.Parse(submission[7]);
            Globals.temp = int.Parse(submission[8]);
            Globals.light = int.Parse(submission[9]);
            Globals.humidity = int.Parse(submission[10]);
            string response = "ACK";
            if (Globals.soil < Globals.desiredwaterlevel)
            {
                response = "motoron";
            }
            else
            {
                response = "motoroff";
            }
            byte[] msg = System.Text.Encoding.ASCII.GetBytes(response); // System.Text.Encoding.ASCII.GetBytes(data);

            // Send back a response.
            stream.Write(msg, 0, msg.Length);
            Console.WriteLine("Sent: {0}", response);
            break;
        }
    }
```

Fig. 4 How Different Methodologies were used to process data from the devices

In Fig. 5, we show the WiFi when a connection between the microcontroller, relay, and algorithms have been established. The figure shows the information that has been gathered from the feedback process.

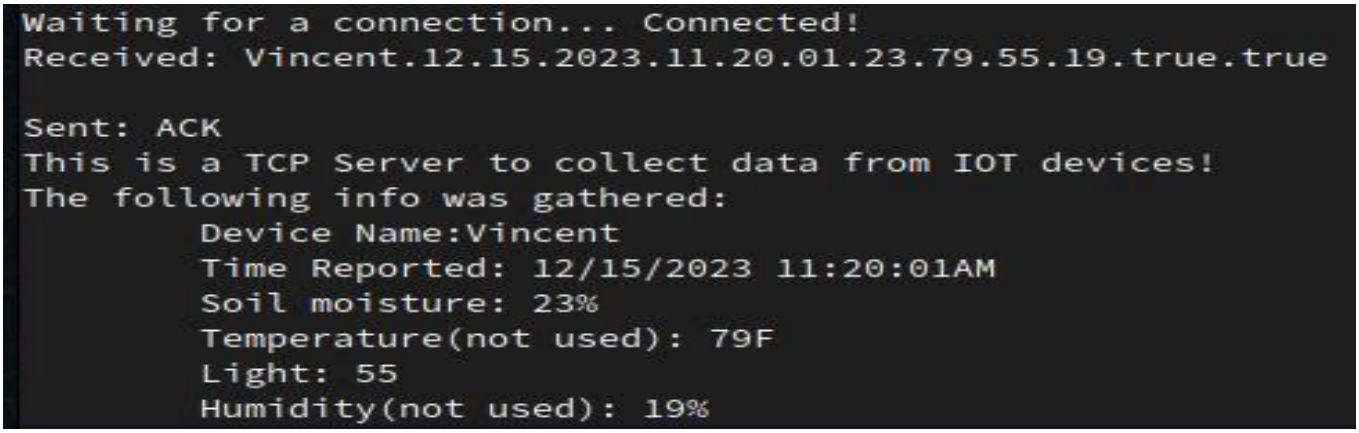

Fig. 5 Information that has been gathered from the feedback process.

All these steps are extremely important because a single mistake can lead to a real damage since the code controls real life things. The *if* statements must be carefully used where appropriate and not to perform the same thing twice, and for processing things like strings, we must figure out the most practical way to parse them.

6. Web Interface Implementation

The web interface has been written using C#. C# interface represents a contract between an object and its user. Interfaces in C# can help us achieve good things allowing objects to interact with each other. C# interface acts as a blueprint for our methods and events. It makes it easier to follow the project. We can make the project come to specific requirements using C# and it will make our code more flexible. C# interface improves extensibility where other developers can extend the program. In the end, C# interfaces are easy to create and implement, and can also be inherited from other interfaces.

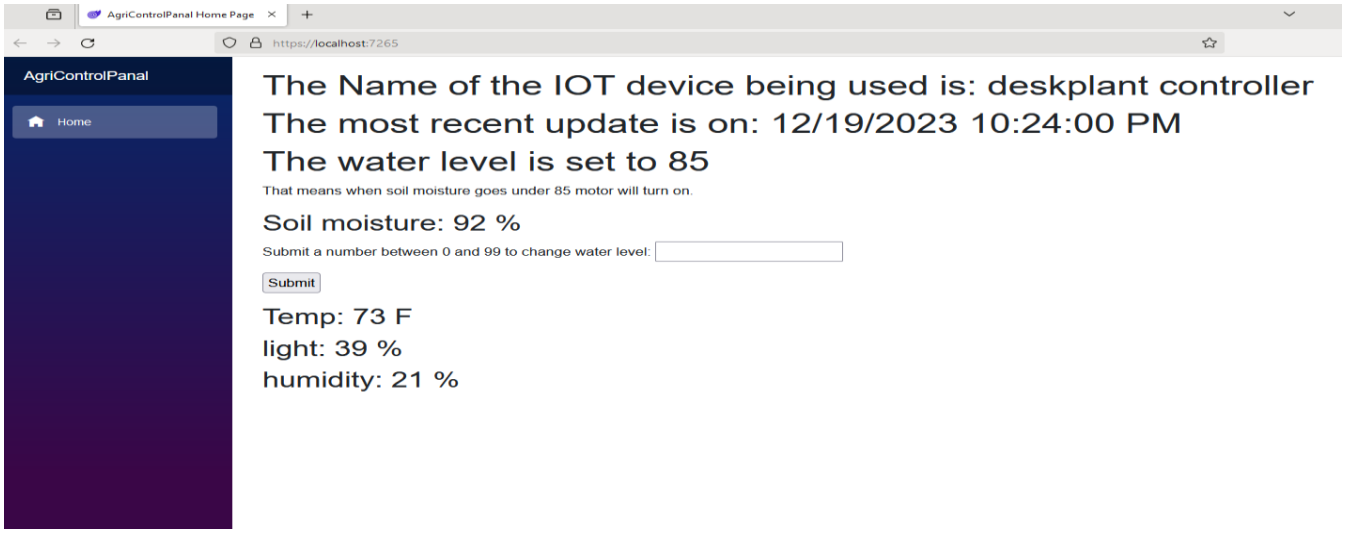

Fig. 6 The Web Page Information

Fig. 6, shows the current interface for controlling the system, it's simple and responsive and contains a text box that allows us to update the watering rate of the plant and see data from all the sensors working together; and Fig. 7 shows the communication information to the rest of the system.

```
@page "/"

<PageTitle>AgriControlPanal Home Page</PageTitle>

<h1>The Name of the IOT device being used is: @iotname</h1>
<h1>The most recent update is on: @Time</h1>
<h1>The water level is set to @Globals.desiredwaterlevel</h1>
<p>That means when soil moisture goes under @Globals.desiredwaterlevel motor will turn on.</p>
<h2>Soil moisture: @soil %</h2>
<form action="" method="get">
<p>
    <label for="waterlevel">Submit a number between 0 and 99 to change water level:</label>
    <input type="text" name="waterlevel" />
    <p><input type="submit" value="Submit" /></p>
</p>
<h2>Temp: @temp F</h2>
<h2>light: @light %</h2>
<h2>humidity: @humidity %</h2>
</form>


@code {
    public void OnGet(int waterlevel) {
        Globals.desiredwaterlevel = waterlevel; // does not work yet
    }

    private string iotname = Globals.iotdevice_name;
    private DateTime Time = Globals.Timestamp;
    private int soil = Globals.soil;
    private int temp = Globals.temp;
    private int light = Globals.light;
    private int humidity = Globals.humidity;
}
```

Fig. 7 Information that shows the way it communicates with the rest of the system.

7. Conclusion

Our Agricultural System was successfully implemented. Throughout the process we learned how to use IoT devices that can make the world more sustainable by helping us make complicated decisions based on many factors from data that is hard to gather on our own. By having an efficient IoT system, we can control it remotely with not much difficulty. Therefore, making IoT devices are very useful for helping us to keep the cost down. These outcomes might open the door to larger applications such as using databases that could handle larger quantities of data. Overall, our solution implementation has shown how useful IoT devices can be and how lots of planning and tinkering can simplify these complicated tasks.

References

[1] ESP32-C6 Board with built in relays by Olimex. (2024, November 17). [Online]. Available: https://www.olimex.com/Products/IoT/ESP32-C6/ESP32-C6-EVB/open-source-hardware

[2] Example Code for TCP over Wifi client for ESP32, (2024, October 12). [Online]. Available: https://github.com/espressif/arduino-esp32/blob/master/libraries/WiFi/examples/WiFiClientBasic/WiFiClientBasic.ino

[3] Reasons to use IOT in Agricultural by SmartTek Solutions, (2024, November, 19). [Online]. Available: https://smarttek.solutions/blog/iot-in-agriculture/

[4] B. Wagner. (2024, November 24). Interface - C# Reference – C#. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/interface

[5] Bala. (2024, December 2). Understanding Interfaces in C#. [Online]. Available: https://www.c-sharpcorner.com/UploadFile/sekarbalag/Interface-In-CSharp/