SMART IRRIGATION SYSTEM

TEAM MEMBERS:

Khushal Goyal | 21UCS109

Nukul Sharma | 21UCS253

The LNM Institute of Information Technology

COURSE NAME:

DPIoT

PERFORMED UNDER:

Rajbir Kaur

Sunil Kumar

Table of Contents

OBJECTIVE

Design a smart irrigation system using IoT to optimize water usage for agriculture. The system should be able to monitor the soil moisture, temperature, and humidity in real-time and take appropriate actions, such as turning on or off the water pump, to maintain the optimal conditions for crop growth. The system should also be able to provide notifications to the user via a mobile app, indicating the status of the irrigation system and any maintenance required. The goal is to reduce water usage, minimize labour costs, and increase crop yield while maintaining sustainable practices.

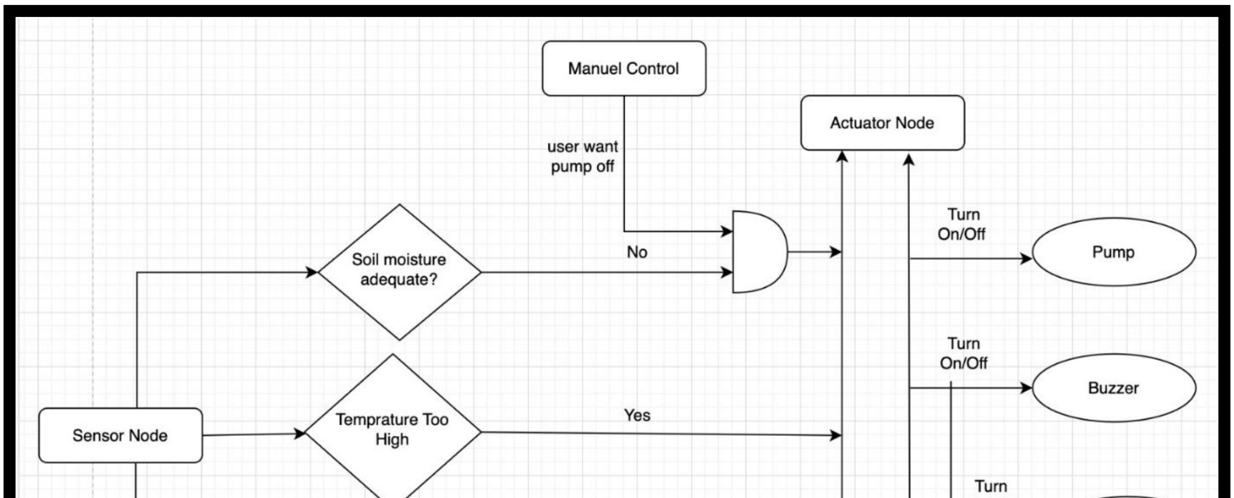
Hardware Used

- **Esp8266 NodeMCU**
- **Jumper wires**
- **Servo motor 9g**
- **DHT22 senor**
- **Buzzer**
- **Soil humidity sensor**

Software Used

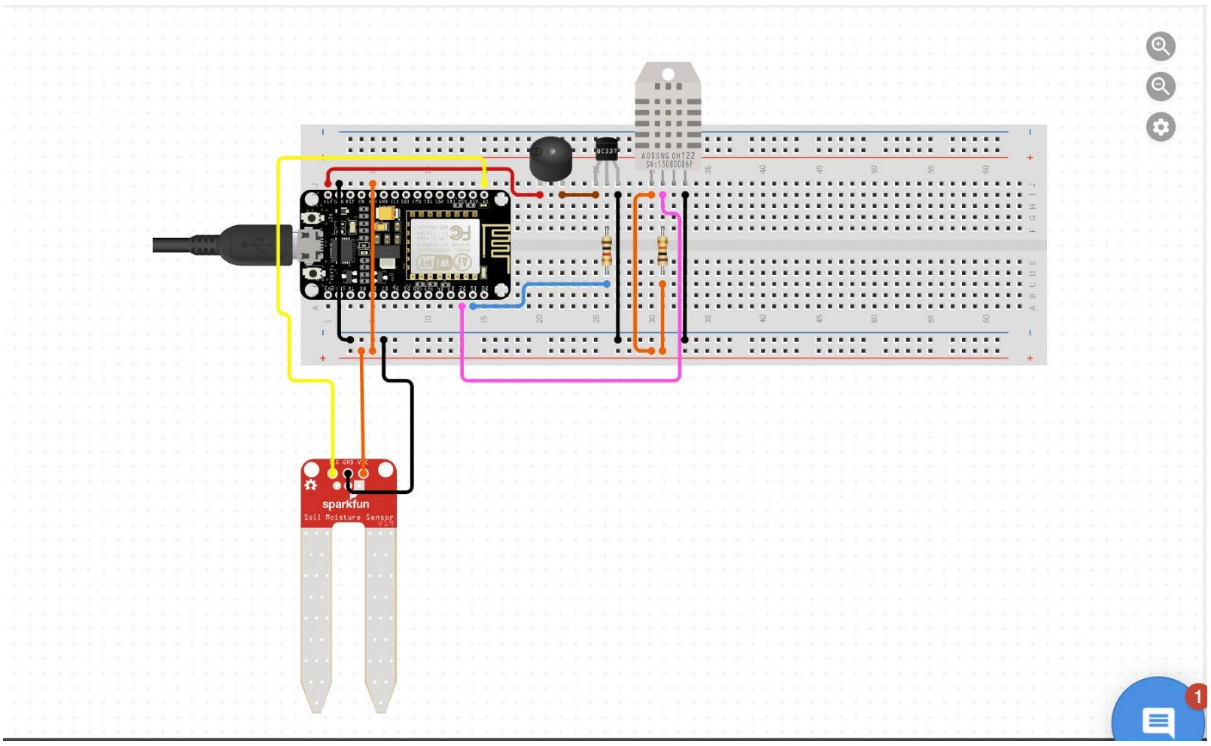
- **Blynk app**
- **Thinkspeak**
- **Arduino ide**

1. DATA-FLOW DIAGRAM

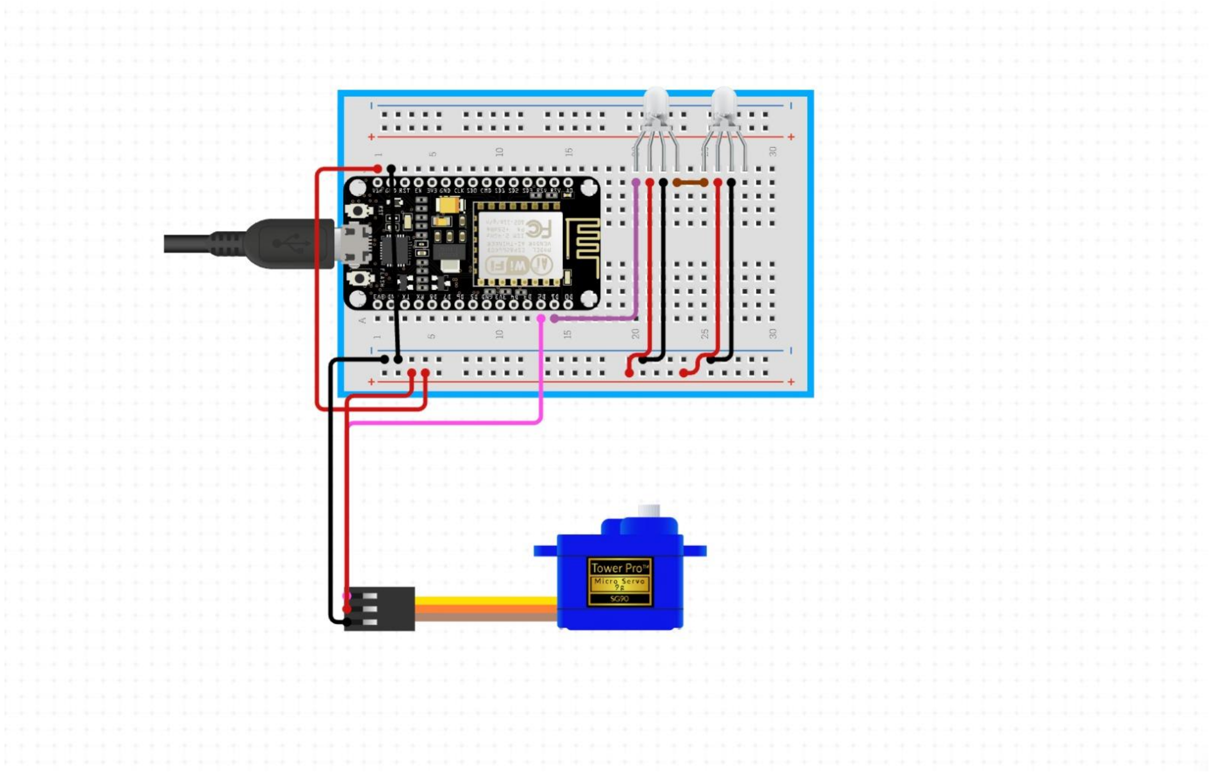


2. CIRCUIT DIAGRAM

2.1. SENSOR NODE



2.2. ACTUATOR NODE



3. IMPLEMENTATION

3.1. Hardware Setup:

a. ESP8266 Node 1 with Soil Humidity Sensor.

b. ESP8266 Node 2 with Water Pump and Green LED and Red LED.

c. DHT22 Temperature and Humidity Sensor.

d. Additional components like resistors, capacitors, led lights, jumper wires, and breadboard.

3.2. Software Setup:

a. Install Arduino IDE and the required libraries like Blynk .

b. Configure both ESP8266 nodes with WiFi SSID and Password and UART Hardware.

c. Configure Blynk app with Auth Token and create two buttons for turning the pump and LED on/off.

3.3. Algorithmic Implementation:

a. Node 1 Algorithm:

i. Connect to Wi-Fi and Blynk app and Thinkspeak.

ii. Read soil humidity value from the sensor.

iii. If soil humidity is below the threshold, publish a message to the TX port to turn on the water pump.

iv. Read temperature and humidity values from the DHT22 sensor.

v. If humidity is above 80, send ‘stop’ message to TX port.

vi. If temperature is above 40 then send ‘hot’ message to TX port.

b. Node 2 Algorithm:

i. Connect to WiFi and Blynk app.

ii. Listen from blynk app.

iii. Listen from the RX and if the message is ‘on’ && user wants the pump on then turn on the green led and motor

iv. If message from RX port is ‘stop’ then stop the pump forcefully.

v. If the message is ‘hot’ then turn on the red led to show

3.4. Deployment:

a. Connect Node 1 and Node 2 to the respective hardware components as per the block diagram.

b. Upload the Node 1 and Node 2 code to the respective ESP8266 nodes.

c. Power on the nodes and verify that the sensors are reading the correct values and the water pump and LED are turning on/off as per the threshold values.

d. Test the system with real-time conditions and fine-tune the threshold values as per the requirement.

4. CONCLUSION

In conclusion, the use of IoT technology in smart irrigation systems has the potential to revolutionize the agriculture industry. By incorporating sensors, controllers, and other technologies, farmers can monitor and optimize their irrigation systems remotely, saving time, water, and money while increasing crop yields. The integration of robotics, drones, and satellite imagery can further enhance the accuracy and efficiency of irrigation management. With the advancement of technology, we can expect even more innovations in this field, leading to more sustainable and productive agriculture practices. Overall, the adoption of smart irrigation systems is a step towards a more sustainable future, and it is an exciting time for the agriculture industry as it embraces these technological advancements.

5. FUTURE SCOPE

As we have made a Smart irrigation prototype, The real project can be enhanced using below mentioned technologies.

5.1. Robotics:

Robotic systems have become increasingly prevalent in agriculture, and smart irrigation systems are no exception. In the future, robotic systems will be used to perform various tasks in smart irrigation systems, including sensing soil moisture levels, monitoring crop growth, and even irrigating crops. With the use of robotic systems, smart irrigation systems will be more precise, efficient, and reduce labour costs.

5.2. Drones:

Drones are another technology that will play a significant role in the future of smart irrigation systems. Drones can be used to gather data on soil moisture levels, crop health, and even apply water and fertilizers precisely. This technology will enable farmers to make informed decisions about when and where to irrigate their crops, thus improving efficiency and reducing water wastage.

5.3. Satellites:

Satellites will also be a critical component of smart irrigation systems in the future. With satellite imagery, farmers will be able to detect patterns in crop growth and water usage across large areas, making it easier to identify areas that require more or less water. This technology will also provide a more comprehensive view of the impact of weather patterns on crops, enabling farmers to make more informed decisions about irrigation.

5.4. Artificial Intelligence:

Artificial intelligence (AI) will play a significant role in the future of smart irrigation systems. With AI, smart irrigation systems will be able to predict when and where irrigation is needed, based on data gathered from sensors, drones, and satellites. This technology will enable farmers to optimize irrigation schedules, reducing water usage and increasing crop yields.

APPENDIX

Sensor Node Code

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <BlynkSimpleEsp8266.h>
#include <WiFiClient.h>
#include <DHT.h>
#include <ThingSpeak.h>

//pins used
//A0 for soil
//D4 for dht
//D1 for buzzer
```

```
#define on 1
#define off 2
#define hot 3
#define cold 4
#define isHumid 5
#define isNotHumid 6
//Blynk Details
#define BLYNK_AUTH_TOKEN "mKZc8vmjzqLRypf21cUN0sV1DeuYluEu"
#define WIFI_SSID "Gryffindor's Tower"
#define WIFI_PASSWORD "qazwsx1234"

const IPAddress receiver_ip(192, 168, 219, 9);
WiFiClient client;

#define DHTPIN 2
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);


int buzzerPin = 5;
float temperature;
float humidity;

int soilPin = A0;
int soilValue = 0;

unsigned long Channel_ID = 2139609;            // Channel ID
const char *WriteAPIKey = "OOF33YSR4CNF3HDV";  // Your write API Key
long delayStart = millis();

void setup() {
  Serial.begin(9600);
  pinMode(buzzerPin, OUTPUT);

  WiFi.disconnect();
  delay(2000);
  Serial.print("Start Connection");  //Starting the fresh Wifi Connection
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(200);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
  // Connect to WiFi network
  ThingSpeak.begin(client);
  Blynk.begin(BLYNK_AUTH_TOKEN, WIFI_SSID, WIFI_PASSWORD);  //Starting the
Blynk App Connection
```

```
  dht.begin();                                                   //initializing the 
DHT sensor for Readings
}

void loop() {
  Blynk.run();
  soilValue = analogRead(soilPin);
  Serial.print("Soil Moisture: ");

  //Read soil value from the sensor
  Serial.println(soilValue);

  //Read temprature and humidity via DHT sensor
  temperature = dht.readTemperature();
  humidity = dht.readHumidity();

  if (isnan(temperature)) {
    Serial.println("Failed to read temperature from DHT sensor!");
  } else {
    Serial.print("Temperature: ");
    Serial.print(temperature);
    Serial.print(" °C");
  }

  if (isnan(humidity)) {
    Serial.println("Failed to read humidity from DHT sensor!");
  } else {
    Serial.print(" | Humidity: ");
    Serial.print(humidity);
    Serial.println(" %");
  }



  if ((millis() - delayStart) >= 15000) {
    ThingSpeak.writeField(Channel_ID, 1, soilValue, WriteAPIKey);
  }
  if ((millis() - delayStart) >= 15000) {
    if (!isnan(temperature))
      ThingSpeak.writeField(Channel_ID, 2, temperature, WriteAPIKey);
  }
  if ((millis() - delayStart) >= 15000) {
    if (!isnan(humidity))
      ThingSpeak.writeField(Channel_ID, 3, humidity, WriteAPIKey);
  }

  Blynk.virtualWrite(V2, soilValue);
  Blynk.virtualWrite(V3, temperature);
  Blynk.virtualWrite(V4, humidity);
  if (temperature > 35) {  //This means that the atmospheric temprature is
above the threashhold so we will do actuation like covering with shead
    if (client.connect(receiver_ip, 80)) {
      client.print(hot);  // Send sensor value to receiver NodeMCU
      // delay(100);
      // client.stop();
    }
  } else {
    if (client.connect(receiver_ip, 80)) {
      client.print(cold);  // Send sensor value to receiver NodeMCU
      // delay(100);
      // client.stop();
    }
  }


  //If the moisture in soil is below the threshold so we will send signal to
the other node to start pump
  if (soilValue > 600) {
    if (client.connect(receiver_ip, 80)) {
      client.print(on);  // Send sensor value to receiver NodeMCU
      // delay(100);
      // client.stop();
    }
  } else {
    if (client.connect(receiver_ip, 80)) {
      client.print(off);  // Send sensor value to receiver NodeMCU
      // delay(100);
      // client.stop();
    }
  }


  if (humidity > 80) {
    if (client.connect(receiver_ip, 80)) {
      client.print(isHumid);  // Send sensor value to receiver NodeMCU
      // client.stop();
    }
  } else {
    if (client.connect(receiver_ip, 80)) {
      client.print(isNotHumid);  // Send sensor value to receiver NodeMCU
      // delay(100);
      // client.stop();
    }
  }
  client.stop();
  // delay(1000);
```

```
}

BLYNK_WRITE(V5) {
  int temp = param.asInt();  // Read the value of temperature from the Blynk
app
  if (temp >= 35) {
    Serial.println("Temprature is too hot. Cover the crops");
    digitalWrite(buzzerPin, HIGH);
  } else {
    digitalWrite(buzzerPin, LOW);
  }

  if (client.connect(receiver_ip, 80)) {
    client.print(hot);  // Send sensor value to receiver NodeMCU
    // delay(100);
    // client.stop();
  } else {
    if (client.connect(receiver_ip, 80)) {
      client.print(cold);  // Send sensor value to receiver NodeMCU
        // delay(100);
        // client.stop();
    }
  }
}
```

Actuator Node Code

```
#define BLYNK_PRINT Serial
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>
#include <Servo.h>
#include <WiFiClient.h>
Servo myservo;
int pos = 0;

//Blynk details
char auth[] = "mKZc8vmjzqLRypf21cUN0sV1DeuYluEu";
char ssid[] = "Gryffindor's Tower";
char pass[] = "qazwsx1234";

//Object for Creating a Wifi - Server
WiFiServer server(80);
```

```
int incomingData;
int ledPin =
2;
// the pin that the LED is attached to
bool ledStatus =
false;                                                                             //
variable to store the status of the LED
int commands[6] = { 1,2,3,4,5,6 };  // Array of possible commands
int commandCount =
6;                                                                               //
Number of elements in the commands array


int tempPin = 5;
int flag = 1;
int humidFlag = 1;


void setup() {
  Serial.begin(9600);  //For Serial Connection to serial Monitor
  pinMode(ledPin, OUTPUT);
  pinMode(tempPin, OUTPUT);

  Serial.print("Connecting to ");  //Connecting to Wifi
  Serial.println(ssid);

  WiFi.begin(ssid, pass);

  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");  //Showing IP Address of Connection
  Serial.println(WiFi.localIP());

  server.begin();  //Starting the Server

  Blynk.begin(auth, ssid, pass);  //Starting the Blynk App Connection
  // Blynk.syncVirtual(V3, V4);            //Syncing Virtual Pins
}

void loop() {
  WiFiClient client = server.available();
  if (client) {
    Blynk.run();                       //Running the Blynk App Client
    while (client.connected()) {  //Checking for client Connection
      if (client.available()) {
        incomingData = client.read() - '0';
        Serial.println(incomingData);
        for (int i = 0; i < commandCount; i++) {  // Loop through the possible
commands
          if (incomingData == commands[i]) {      // If the incoming data
matches the current command
            executeCommand(i);                    // Call the function to
execute the command
            break;                                // Exit the loop
          }
        }  //Reading the Data Sent by Client
        incomingData = -1;
      }
    }
  }

  // while (Serial.available() > 0) {               // Check if there is any
data available on the serial port
  //   char incomingChar = Serial.read();           // Read the incoming data
  //   incomingData += incomingChar;                // Append the incoming
character to the incomingData string
  //   if (incomingChar == '\n') {                  // If a newline character
is received
  //     for (int i = 0; i < commandCount; i++) {   // Loop through the
possible commands
  //       if (incomingData == commands[i]) {       // If the incoming data
matches the current command
  //         executeCommand(i);                     // Call the function to
execute the command
  //         break;                                 // Exit the loop
  //       }
  //     }
  //     incomingData = "";  // Reset the incomingData string
  //   }
  // }
  client.stop();
}

void executeCommand(int commandIndex) {
  switch (commandIndex) {
    case 0:  // If the command is "on"
      if (flag) {
        if (humidFlag) {
          digitalWrite(ledPin, HIGH);
          myservo.attach(D2);
        }
      }
      // Turn on the LED
      break;
    case 1:  // If the command is "off"
      // digitalWrite(ledPin, LOW);    // Turn off the LED
      digitalWrite(ledPin, LOW);
      myservo.detach();
      break;
    case 2:  // If the command is "hot"
      Serial.println("Atmospheric Temprature is very hot cover the crops with shed");
      break;
    case 3:  // if command is "cold"
      Serial.println("Atmospheric Temprature is appropriate rest assured!!");
      break;
    case 4:  // if command is "isHumid"
      humidFlag = 0;
      myservo.detach();
            digitalWrite(ledPin, LOW);
      break;
    case 5:  // if command is "isNotHumid"
      humidFlag = 1;
      break;
  }
}

BLYNK_WRITE(V1)  // this function gets called whenever the button widget is turned on or off
{
  int buttonState = param.asInt();  // get the state of the button

  if (buttonState == 1)  // if the button is turned on
  {
    digitalWrite(ledPin, HIGH);  // turn on the LED
    ledStatus = true;            // update the LED status variable
    myservo.attach(D2);
    flag = 1;
  } else  // if the button is turned off
  {
    digitalWrite(ledPin, LOW);  // turn off the LED
    ledStatus = false;          // update the LED status variable
    myservo.detach();
    flag = 0;
  }
}


BLYNK_WRITE(V5) {
  int temp = param.asInt();  // Read the value of temperature from the Blynk app
  if (temp >= 35) {
    Serial.println("Temprature is too hot. Cover the crops");
    digitalWrite(tempPin, HIGH);
  } else {
    digitalWrite(tempPin, LOW);
  }
}
```

Bibliography

- Adafruit Learning System. (n.d.). UART (Serial) Communication Between Two ESP8266-01/01S Modules. Retrieved from https://learn.adafruit.com/uart-serial-comm-between-two-esp8266-modules
- Electronics Hub. (2018). DHT22 Sensor with ESP8266 and Arduino Uno. Retrieved from https://www.electronicshub.org/dht22-sensor-arduino-uno-esp8266/
- Maker Pro. (2019). Getting Started with ESP8266 and DHT22 Sensor. Retrieved from https://maker.pro/esp8266/tutorial/getting-started-with-esp8266-and-dht22-sensor
- Random Nerd Tutorials. (n.d.). ESP32 with DHT11/DHT22 Temperature and Humidity Sensor using Arduino IDE. Retrieved from https://randomnerdtutorials.com/esp32-dht11-dht22-temperature-humidity-sensor-arduino-ide/
- ThingSpeak. (n.d.). ThingSpeak Communication Library for Arduino, ESP8266 and ESP32. Retrieved from https://www.mathworks.com/help/thingspeak/arduino-and-thingSpeak.html

- TutorialsPoint. (n.d.). NodeMCU - Connecting with DHT22. Retrieved from https://www.tutorialspoint.com/nodemcu/nodemcu_connecting_with_dht22.htm
- UART Protocol. (n.d.). UART Protocol Basics. Retrieved from https://www.electronicshub.org/uart-protocol-basics/