A Theme Based Project Report

on

DELIGHTFUL DRIPS : A SMART IRRIGATION SYSTEM

Submitted in partial fulfilment of the
Requirements for the award of the Degree of
BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE & ENGINEERING

By

Akash S Vora

1602-19-733-126

Bakshi Abhinith

1602-19-733-124

Department of Computer Science & Engineering

Vasavi College of Engineering (Autonomous)

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad-31

2022

Vasavi College of Engineering (Autonomous)

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Computer Science & Engineering

DECLARATION BY THE CANDIDATE

We, **AKASH S VORA** and **BAKSHI ABHINITH,** bearing hall ticket numbers **1602-19-733-126** and **1602-19-733-124** respectively, hereby declare that the project report entitled **"DELIGHTFUL DRIPS: A SMART IRRIGATION SYSTEM"** Department of Computer Science & Engineering, VCE, Hyderabad, is submitted in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering** in **Computer Science & Engineering**.

This is a record of bonafide work carried out by me and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

AKASH S VORA,

1602-19-733-126.

BAKSHI ABHINITH,

1602-19-733-124.

Vasavi College of Engineering (Autonomous)

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Computer Science & Engineering

BONAFIDE CERTIFICATE

This is to certify that the project entitled **"DELIGHTFUL DRIPS: A SMART IRRIGATION SYSTEM"** being submitted by **AKASH S VORA** and **BAKSHI ABHINITH,** bearing hall ticket numbers **1602-19-733-126** and **1602-19-733-124** respectively, in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science & Engineering is a record of bonafide work carried out by him/her under my guidance.

Mr. P. Narsaiah

Assistant Professor

Dept. of CSE

ACKNOWLEDGEMENT

With immense pleasure, we record our deep sense of gratitude to our guide Mr. P. Narsaiah, Assistant Professor, Vasavi College of Engineering, Hyderabad, for the valuable guidance and suggestions, keen interest and thorough encouragement extended throughout the period of the project work. I consider myself lucky enough to be part of this project. This project would add as an asset to my academic profile.

We express our thanks to all those who contributed for the successful completion of our project work.

ABSTRACT

India is the country of villages and agriculture plays an important role for the development of the country. Water is an essential resource in agriculture and its optimal management plays a crucial role.

Irrigation system is a method of allowing water to drip slowly to the roots of plants, either onto the soil surface or directly onto the root zone, through solenoid valve. However, it is found that the market price of the system is expensive for small area coverage.

Thus, our aim is to devise a cost effective, smart irrigation system that operates based on the real-time moisture content in the soil. This project uses a master slave architecture where each NodeMCU ESP8266 acts as a slave and the soil moisture information collected by each sensor is finally sent to the Raspberry Pi, which acts as a master. The Raspberry Pi controls the solenoid valve and the water is sprinkled if the moisture content is less than 30% in the soil.

TABLE OF CONTENTS

Page No.

1. LIST OF FIGURES

2. INTRODUCTION

Irrigation system is a method of allowing water to drip slowly to the roots of plants, either onto the soil surface or directly onto the root zone, through solenoid valve.

Appropriate soil moisture level is a necessary pre-requisite for optimum plant growth. Also, water being an essential element for life sustenance, there is the necessity to avoid its undue usage. Irrigation is a dominant consumer of water. This calls for the need to regulate water supply for irrigation purposes.

2.1 OVERVIEW

First, we connect a soil moisture sensor to each NodeMCU. Then we use the MQTT protocol to establish communication between the NodeMCU ESP8266 and the Raspberry Pi. MQTT uses a publisher subscriber model to transfer messages. Here each NodeMCU acts as a publisher and publishes the soil moisture content as a topic. The Raspberry Pi subscribes to all the topics in order to receive the soil moisture content from each NodeMCU.

The data provided by each NodeMCU is dynamically rendered onto the website. The data is sent to a javascript file, which is eventually overwritten every 2 seconds. This js file is linked to the HTML file, which contains 4 gauges to display the data. Next, we connect the Solenoid Valve to the Raspberry Pi through a 5V relay in order to control the water flow (for the data provided by NodeMCU 1). Water is sprinkled if the moisture content is less than 30% in the soil.

2.2 METHODOLOGY

In this project, we have used the MQTT protocol for communicating between the Raspberry Pi and NodeMCU ESP8266. After receving the information about soil moisture from each sensor, the Raspberry Pi checks whether the soil moisture content is less than 30% or not. If it is less than 30%, then the Raspberry Pi controls and the solenoid valve and sprinkles water onto the soil for approximately 10 seconds. The data about the soil moisture content is dynamically rendered in the user defined HTML page containing 4 gauges. The data is updated in a javascript file, which is linked with the HTML file, and the website refreshes every 2 seconds for accurate information. This data is also sent to the Adafruit.io website so the data can be viewed from anywhere in the world. In addition to this, Adafruit.io provides beautiful dashboards for the display of the data. The solenoid valve can also be controlled using the Blynk App.

➔ ABOUT MQTT:

The MQTT protocol is the de-facto standard for IoT messaging. Standardized by OASIS and ISO, MQTT publish/subscribe protocol provides a scalable and reliable way to connect devices over the Internet. Today, MQTT is used by many companies to connect millions of devices to the Internet.

- **Advantages of MQTT:**

  - ✓ It requires minimal resources since it is lightweight and efficient
  - ✓ Support bi-directional messaging between device and cloud
  - ✓ Can scale to millions of connected devices

- **MQTT Client:**

  MQTT clients publish a message to an MQTT broker and other MQTT clients subscribe to messages they want to receive. Implementations of MQTT clients typically require a minimal footprint so are well suited for deployment on small constrained devices and are very efficient in their bandwidth requirements.

- **MQTT Broker:**

  MQTT brokers receive published messages and dispatch the message to the subscribing MQTT clients. An MQTT message contains a message topic that MQTT clients subscribe to and MQTT brokers use these subscription lists for determining the MQTT clients to receive the message.

- **Quality of Service Levels:**

  MQTT implements 3 quality of service levels for agreement between the sender and receiver: 1) At most once, 2) At least once (1), and 3) Exactly once. These

QoS levels allow for more reliable IoT applications since the underlying messaging infrastructure and adapt to unreliable network conditions.

- **Persistent Sessions:**

MQTT allows for a persistent session between the client and the broker. This allows for sessions to persist even if the network is disconnected. Once the network is reconnected, the information to reconnect the client to the broker still exists.

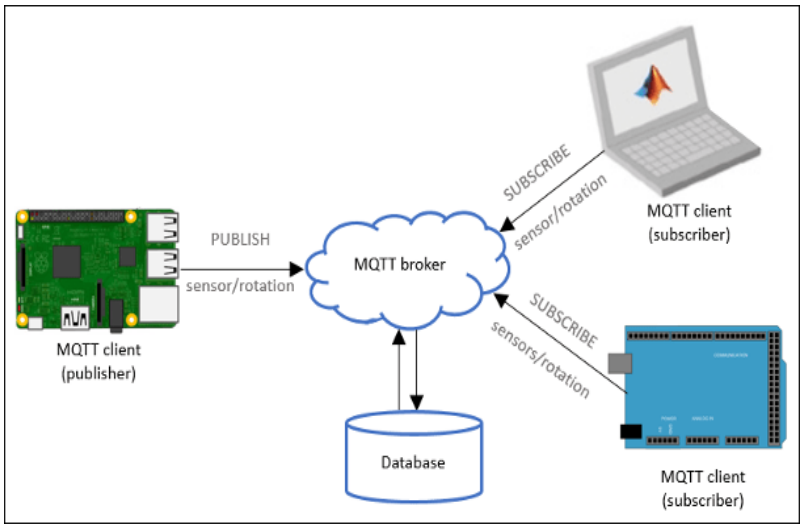


Figure 2.2.1

➔ ABOUT MASTER-SLAVE ARCHITECTURE OF IoT:

Master/slave is a model of communication for hardware devices where one device has a unidirectional control over one or more devices. This is often used in the electronic hardware space where one device acts as the controller, whereas the other devices are the ones being controlled. In short, one is the master and the others are slaves to be controlled by the master.

In electronic technology, it is often used to simplify communication like, instead of having a separate interface to communicate with each disk drive, we can connect most of them via one interface and cable and the computer only has to communicate with one drive serving as the master, then any control command is simply propagated down to the slaves from the master.

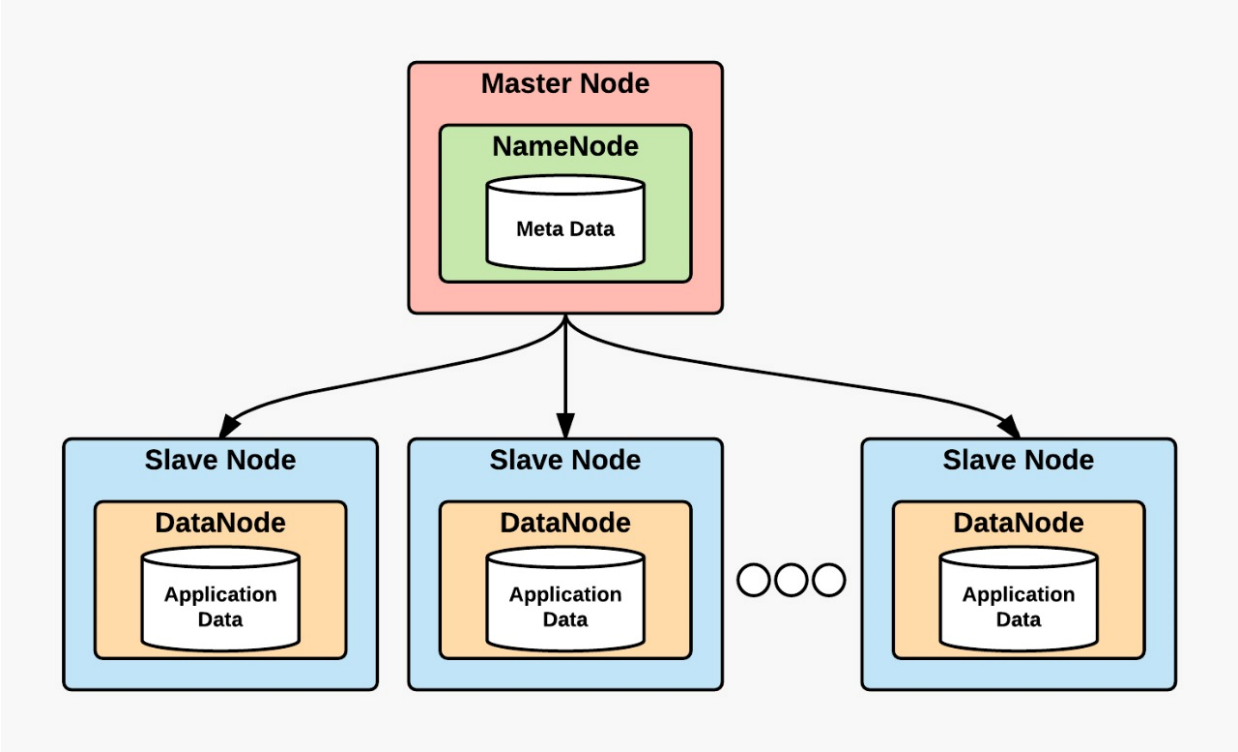


Figure 2.2.2

➔ ABOUT ADAFRUIT.IO:

Adafruit IO is a platform designed to display, respond, and interact with your project's data. It also keeps our data private (data feeds are private by default) and secure. It's the internet of things - for everyone!

Some features of Adafruit IO are :

- Powerful API
- Easy to Use
- Beautiful Dashboards
- Private & Secure

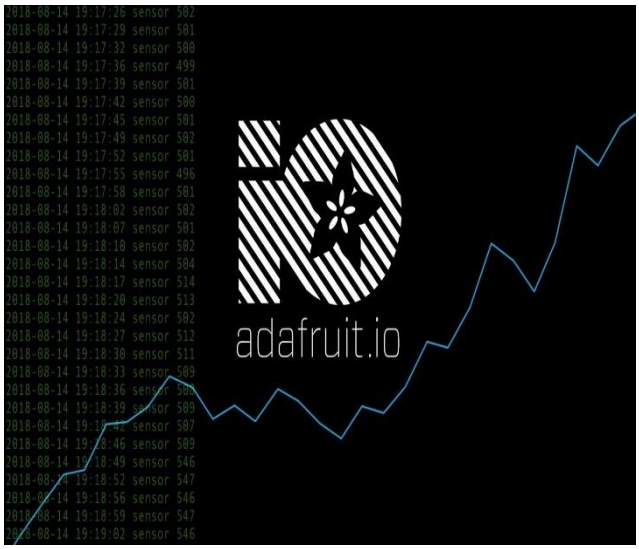


Figure 2.2.3

➔ ABOUT BLYNK:

With Blynk, we can create smartphone applications that allow us to easily interact with microcontrollers or even full computers such as the Raspberry Pi.
The main focus of the Blynk platform is to make it super-easy to develop the mobile phone applications. Developing a mobile app that can talk to Arduino is as easy as dragging a widget and configuring a pin.

Figure 2.2.4

➔ IMPORTANCE TO THE SOCIETY:

This project is important to the society in many aspects.

- Prevents wastage of water
- Healthier crops in the long run
- Prevents over watering of plants.

➔ COMPONENTS USED :

1. RASPBERRY PI

Raspberry Pi is the name of a series of single-board computers made by the Raspberry Pi Foundation. All over the world, people use the Raspberry Pi to learn programming skills, build hardware projects, do home automation etc., It is a very cheap computer that runs Linux, but it also provides a set of GPIO (general purpose input/output) pins, allowing us to control electronic components for physical computing and explore the Internet of Things (IoT).

Figure 2.2.5

2. NODEMCU ESP8266

The NodeMCU (*N*ode *M*icro*C*ontroller *U*nit) is an open-source software and hardware development environment built around an inexpensive System-on-a-Chip (SoC) called the ESP8266. The ESP8266, designed and manufactured by Espressif Systems, contains the crucial elements of a computer: CPU, RAM, networking (WiFi), and even a modern operating system and SDK. That makes it an excellent choice for Internet of Things (IoT) projects of all kinds.

Figure 2.2.6

3. SOIL MOISTURE SENSOR

The moisture of the soil plays an essential role in the irrigation field as well as in gardens for plants.The soil moisture sensor is one kind of sensor used to gauge the volumetric content of water within the soil.

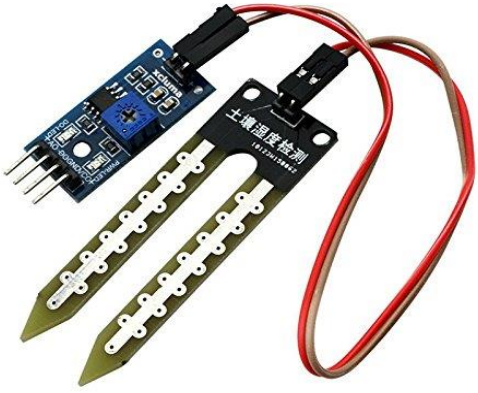


Figure 2.2.7

4. 5V RELAY

Relay is one kind of electro-mechanical component that functions as a switch. The relay coil is energized by DC so that contact switches can be opened or closed. A single channel 5V relay module generally includes a coil, and two contacts like normally open (NO) and normally closed (NC).

Figure 2.2.8

5. SOLENOID VALVE

Solenoid valves are control units which, when electrically energized or de-energized, either shut off or allow fluid flow. The actuator takes the form of an electromagnet. When energized, a magnetic field builds up which pulls a plunger or pivoted armature against the action of a spring.

Figure 2.2.9

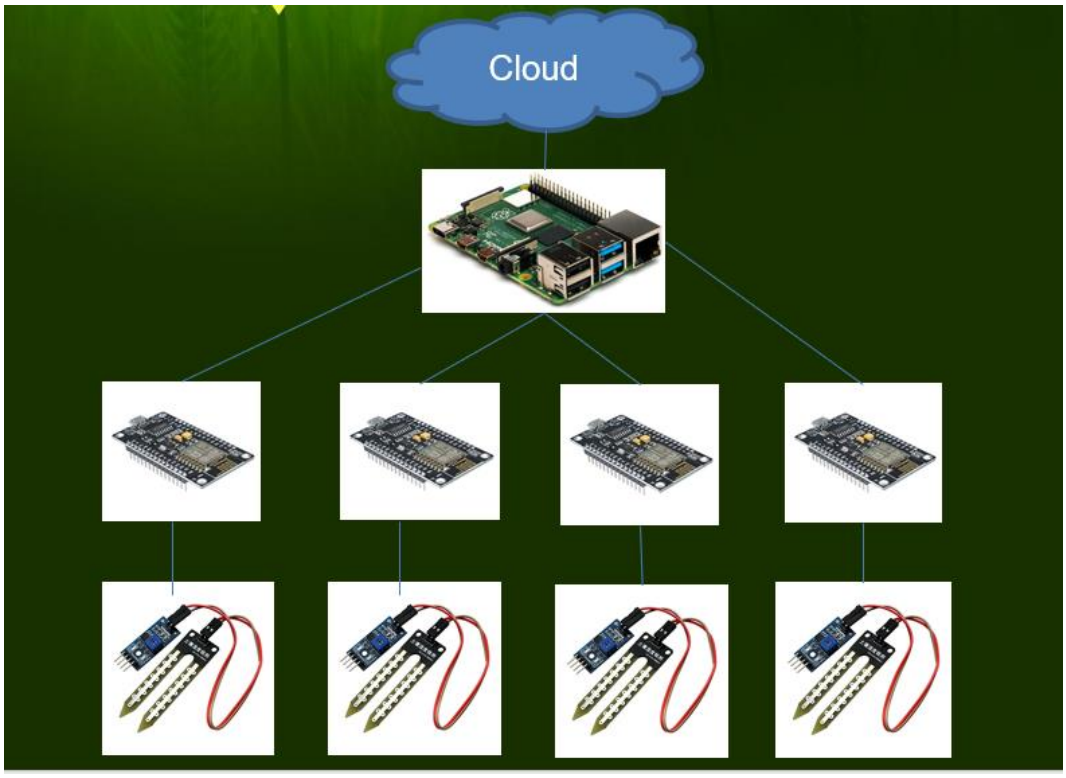
2.3 PROBLEM STATEMENT

To devise a cost effective, smart irrigation system that operates based on the real-time moisture content in the soil.

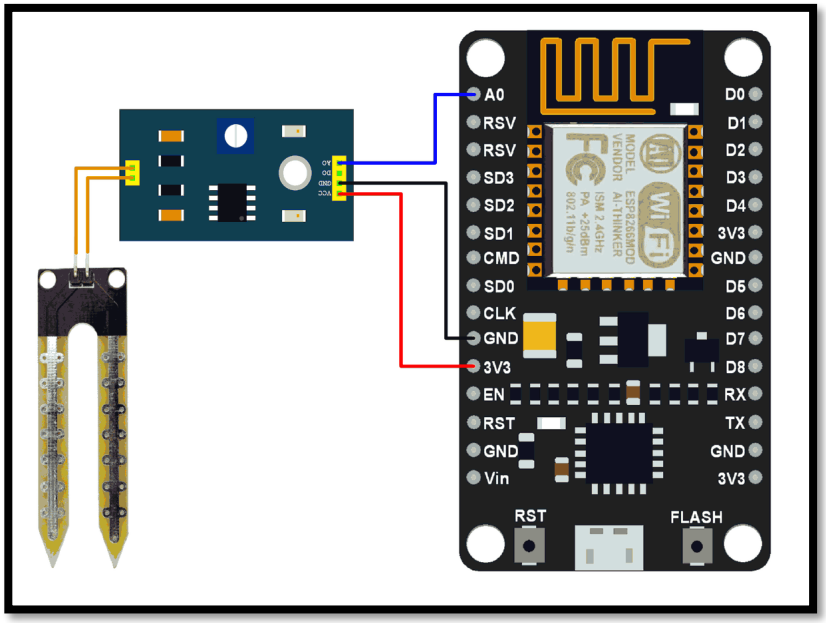
2.4 PROPOSED SOLUTION

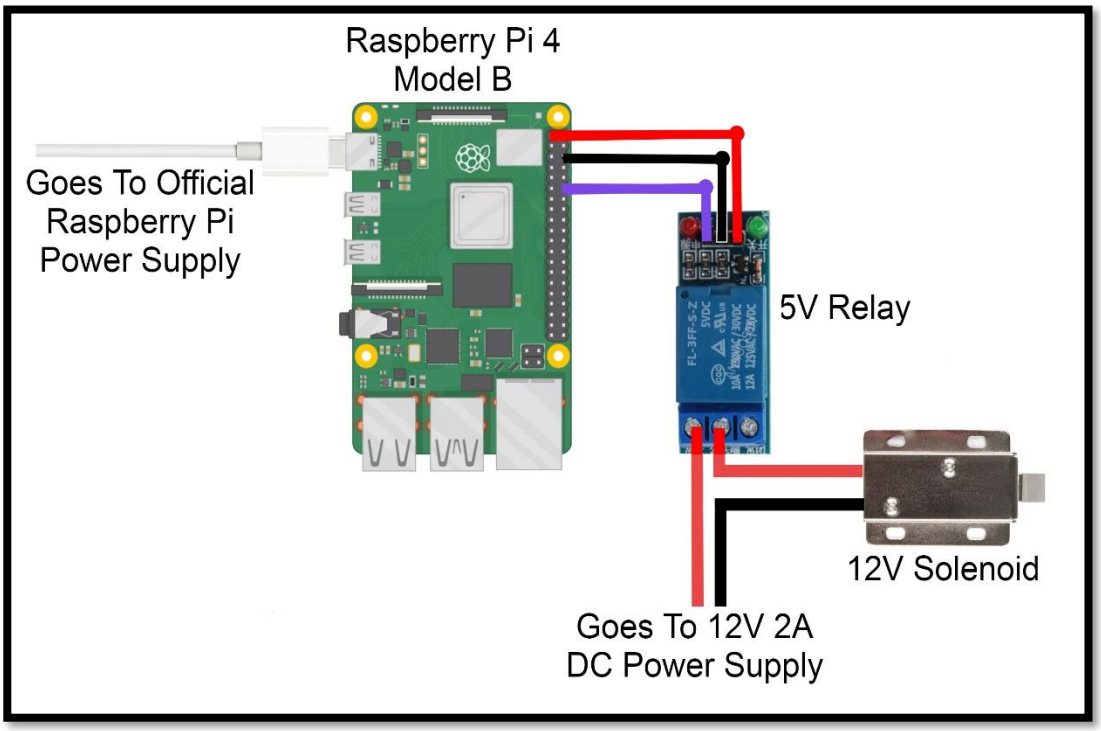
This system installs low cost sensors to sense variables of interest such as soil moisture (as in this case). The data obtained can be monitored using a mobile/PC. If the obtained soil moisture content fails to acquire the minimum criteria, then the field is automatically irrigated using the solenoid valve.

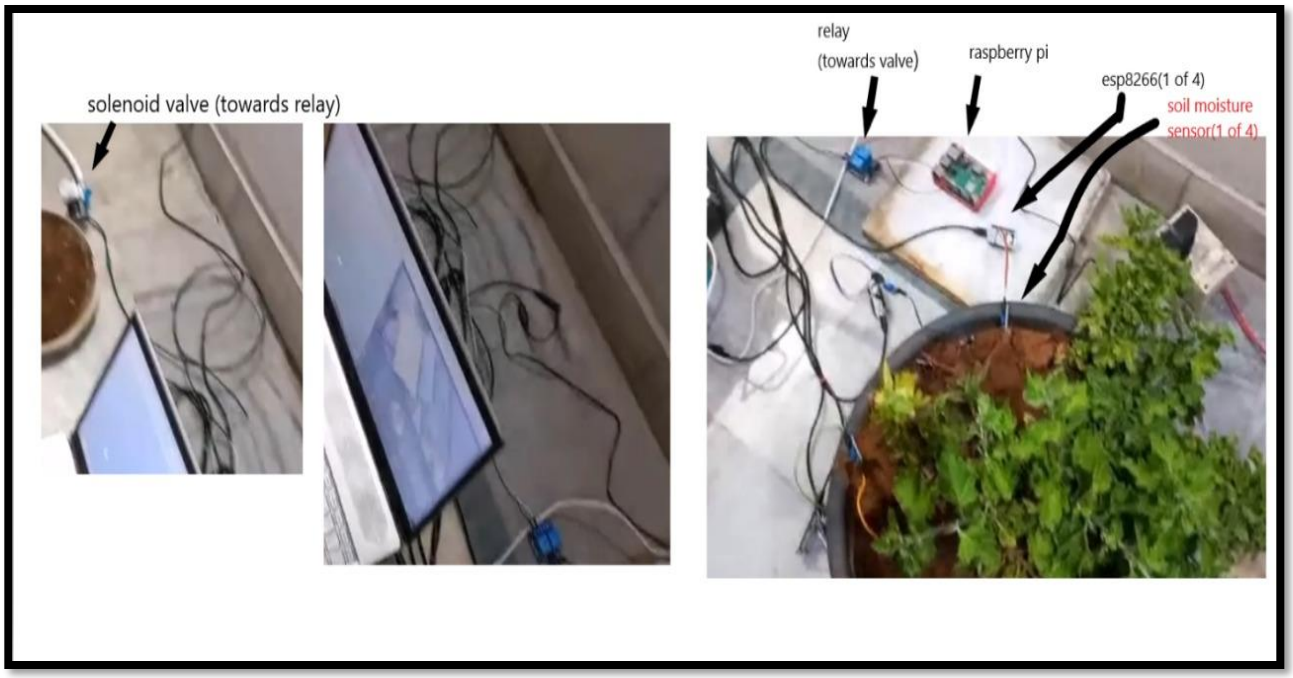
2.5 ARCHITECTURE DIAGRAM



2.6 CIRCUIT DIAGRAMS






solenoid valve (towards relay)
relay
(towards valve)
raspberry pi
esp8266(1 of 4)
soil moisture
sensor(1 of 4)

.

3. SYSTEM REQUIREMENTS

Hardware:

- Raspberry Pi
- NodeMCU ESP8266
- Soil Moisture Sensor
- 5V Relay
- 12V Solenoid Valve
- 12V DC Adaptor
- Input devices: Mouse, Keyboard
- Output devices: Monitor

Software:

- Chrome 76.0 or above
- Windows 7 or above

4. IMPLEMENTATION

Publisher.ino

```
#include <ESP8266WiFi.h> // Enables the ESP8266 to connect to the local network (via WiFi)
#include <PubSubClient.h> // Allows us to connect to, and publish to the MQTT broker

const int sensor_pin = A0;
// WiFi
// Make sure to update this for your own WiFi network!
const char* ssid = "**********";
const char* wifi_password = "**********";

// MQTT
// Make sure to update this for your own MQTT Broker!
const char* mqtt_server = "192.168.0.101";
const char* mqtt_topic = "MCU4";
const char* mqtt_username = "admin";
const char* mqtt_password = "admin";
// The client id identifies the ESP8266 device. Think of it a bit like a hostname (Or just a name, like Greg).
const char* clientID = "MCU4";
// Initialise the WiFi and MQTT Client objects
WiFiClient wifiClient;
PubSubClient client(mqtt_server, 1883, wifiClient); // 1883 is the listener port for the Broker

void setup() {

  Serial.begin(9600);
```

```
  Serial.print("Connecting to ");
  Serial.println(ssid);

  // Connect to the WiFi
  WiFi.begin(ssid, wifi_password);

  // Wait until the connection has been confirmed before continuing
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  // Debugging - Output the IP Address of the ESP8266
  Serial.println("WiFi connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  // Connect to MQTT Broker
  // client.connect returns a boolean value to let us know if the connection was successful.
  // If the connection is failing, make sure you are using the correct MQTT Username and
Password (Setup Earlier in the Instructable)
  if (client.connect(clientID, mqtt_username, mqtt_password)) {
    Serial.println("Connected to MQTT Broker!");
  }
  else {
    Serial.println("Connection to MQTT Broker failed...");
  }
  pinMode(sensor_pin, OUTPUT);

}

void loop() {
```

```
 float moisture_percentage;
 moisture_percentage = ( 100.00 - ( (analogRead(sensor_pin)/1023.00) * 100.00 ) );
 Serial.print("Soil Moisture(in Percentage) = ");
 Serial.print(moisture_percentage);
 Serial.println("%");
 //String mp = String(moisture_percentage);
 char* mp=NULL;
 int len = asprintf(&mp, "%g", moisture_percentage);
 if (len == -1){
  fprintf(stderr, "Error converting float: %m\n");}
 else{
   printf("float is %s\n", mp);}
 if (client.publish(mqtt_topic, mp)) {
   Serial.println("message sent!");
 }
 // Again, client.publish will return a boolean value depending on whether it succeded or not.
 // If the message failed to send, we will try again, as the connection may have broken.
 else {
   Serial.println("Message failed to send. Reconnecting to MQTT Broker and trying again");
   client.connect(clientID, mqtt_username, mqtt_password);
   delay(10); // This delay ensures that client.publish doesn't clash with the client.connect call
   client.publish(mqtt_topic, mp);
 }
 delay(2000);
}
```

Subscriber.py

```
import paho.mqtt.client as mqtt
import sys
import time
import RPi.GPIO as GPIO
import asyncio
GPIO.setmode(GPIO.BOARD)
GPIO.setup(13,GPIO.OUT)
GPIO.setup(15,GPIO.OUT)
GPIO.setup(16,GPIO.OUT)
GPIO.setup(18,GPIO.OUT)
GPIO.setup(22,GPIO.OUT)
GPIO.output(13,0)
GPIO.output(15,0)
GPIO.output(16,0)
GPIO.output(18,0)
#solenoid
GPIO.output(22,1)
GPIO.setwarnings(False)
'''from Adafruit_IO import RequestError,Client,Feed
username = "******************"
key = "**********************"
aio = Client(username,key)
try:
    test = aio.feeds('mcu1')
    test1 = aio.feeds('mcu2')
    test2 = aio.feeds('mcu3')
    test3 = aio.feeds('mcu4')
except RequestError:
    test_feed = Feed(name = 'mcu1')
```

```
    test_feed = aio.create_feed(test_feed)

    test_feed1 = Feed(name = 'mcu2')
    test_feed1 = aio.create_feed(test_feed1)

    test_feed2 = Feed(name = 'mcu3')
    test_feed2 = aio.create_feed(test_feed2)

    test_feed3 = Feed(name = 'mcu4')
    test_feed3 = aio.create_feed(test_feed3)'"
# Don't forget to change the variables for the MQTT broker!

mqtt_username = "admin"
mqtt_password = "admin"
mqtt_topic = "MCU1"
mqtt_topic1 = "MCU2"
mqtt_topic2 = "MCU3"
mqtt_topic3 = "MCU4"
mqtt_broker_ip = "192.168.0.101"

client = mqtt.Client()
# Set the username and password for the MQTT client
client.username_pw_set(mqtt_username, mqtt_password)

v1,v2,v3,v4 = 0,0,0,0

def renderWebpage(tpc,val):
    global v1
    global v2
    global v3
    global v4
    #path = '/var/www/html/Gauges'
```

```
print('renderrrr')
f = open('/var/www/html/Gauges/guagedemo.js', 'w')
file = '''
        const gaugeElement1 = document.querySelector(".gauge1");
        const gaugeElement2 = document.querySelector(".gauge2");
        const gaugeElement3 = document.querySelector(".gauge3");
        const gaugeElement4 = document.querySelector(".gauge4");

        function setGaugeValue(gauge, value) {
          if (value < 0 || value > 1) {
            return;
          }

          gauge.querySelector(".gauge__fill").style.transform = `rotate(${
            value / 2
          }turn)`;
          gauge.querySelector(".gauge__cover").textContent = `${
            value * 100}%`;
        }

        setGaugeValue(gaugeElement1,"'+str(v1)+'");
        setGaugeValue(gaugeElement2,"'+str(v2)+'");
        setGaugeValue(gaugeElement3,"'+str(v3)+'");
        setGaugeValue(gaugeElement4,"'+str(v4)+'");
        '''

f.write(file)
f.close()
time.sleep(1)
```

```
def on_connect(client, userdata, flags, rc):

    client.subscribe(mqtt_topic)
    client.subscribe(mqtt_topic1)
    client.subscribe(mqtt_topic2)
    client.subscribe(mqtt_topic3)
    print("Subscribed")
    print(client)
    print()

def on_message(client, userdata, msg):
    #setZero()
    print("mess")
    global v1
    global v2
    global v3
    global v4
    print("Topic: ", msg.topic + "\nMessage: " + str(msg.payload))

    print("type",msg.topic[len(msg.topic)-1])
    top = msg.topic[len(msg.topic)-1]
    print("type p",type(msg.payload))
    dec = float(msg.payload.decode('utf-8'))
    if(dec > 0):
        trim = round(dec/100,2)
    else:
        trim = 0
    print("top : ",top)
    print("trim : ",trim)
    if(top == '1'):
        v1 = trim
    elif(top == '2'):
```

```
        v2 = trim
    elif(top == '3'):
        v3 = trim
    else:
        v4 = trim
    print(v1," ",v2," ",v3," ",v4)
    '''aio.send_data(test.key,v1*100)    //Uncomment for sending data to Adafruit.io
    aio.send_data(test1.key,v2*100)
    aio.send_data(test2.key,v3*100)
    aio.send_data(test3.key,v4*100)'''

    print('rend ',v1,v2,v3,v4)
    ledCheck()
    renderWebpage(top,trim)


def setZero():
    GPIO.output(13,0)
    GPIO.output(15,0)
    GPIO.output(16,0)
    GPIO.output(18,0)


def ledCheck():
    global v1
    global v2
    global v3
    global v4
    #print('lckeh')
    print('lch  ',v1,v2,v3,v4)
    if(v1 <= 0.25):
        GPIO.output(13,1)
```

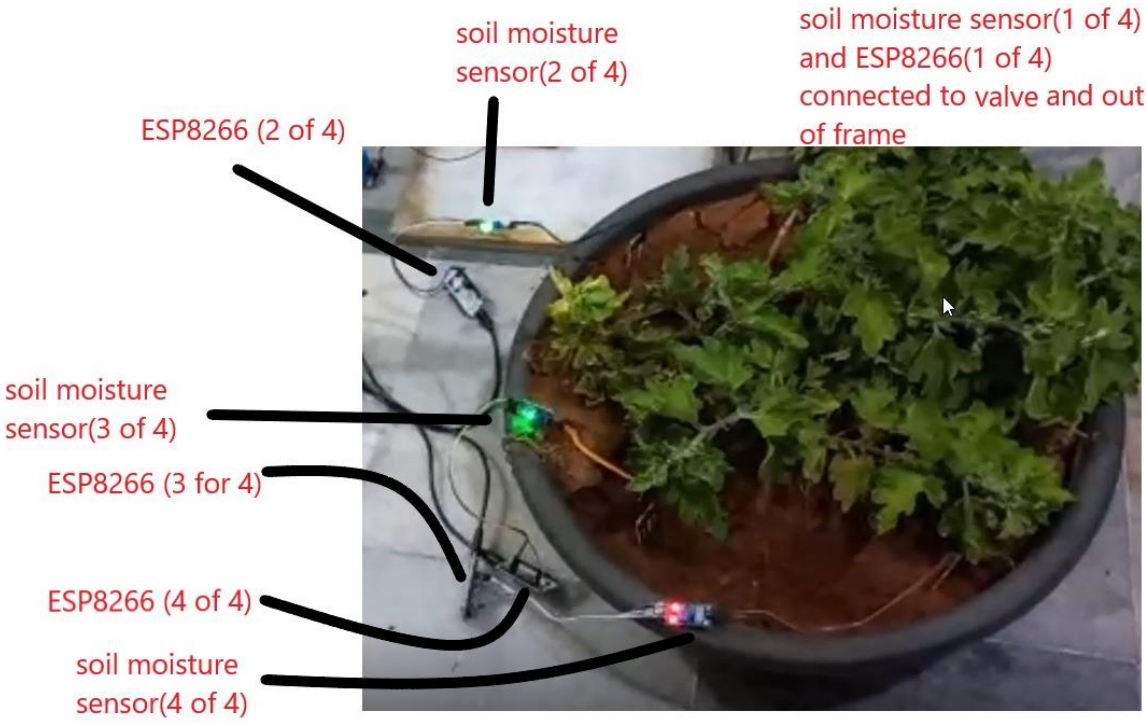
```
    GPIO.output(22,0)
    print('r on')
    time.sleep(3)
    GPIO.output(22,1)
    print('r off')
    #GPIO.output(13,0)
    #time.sleep(1)
    #print('helo')
else:
    GPIO.output(13,0)
    #print('hel')

if(v2 <= 0.30):
    GPIO.output(15,1)
    #time.sleep(1)
else:
    GPIO.output(15,0)

if(v3 <= 0.25):
    GPIO.output(16,1)
    #time.sleep(1)
else:
    GPIO.output(16,0)

if(v4 <= 0.25):
    GPIO.output(18,1)
    #time.sleep(1)
else:
    GPIO.output(18,0)
time.sleep(1)
#setZero()
```

```
#setZero()
client.on_connect = on_connect
client.on_message = on_message

client.connect(mqtt_broker_ip, 1883)

client.loop_forever()
client.disconnect()
```

Publisher_with_Blynk.ino

```
#define BLYNK_PRINT Serial
#include<BlynkSimpleEsp8266.h>

char auth[] = "**********************";
char ssid[] = "******************";
char pass[] = "*****************";

BlynkTimer timer;
void setup() {
  Blynk.begin(auth,ssid,pass);
}
void loop() {
  Blynk.run();
  timer.run();
}
```

5. OUTPUT SCREENSHOTS

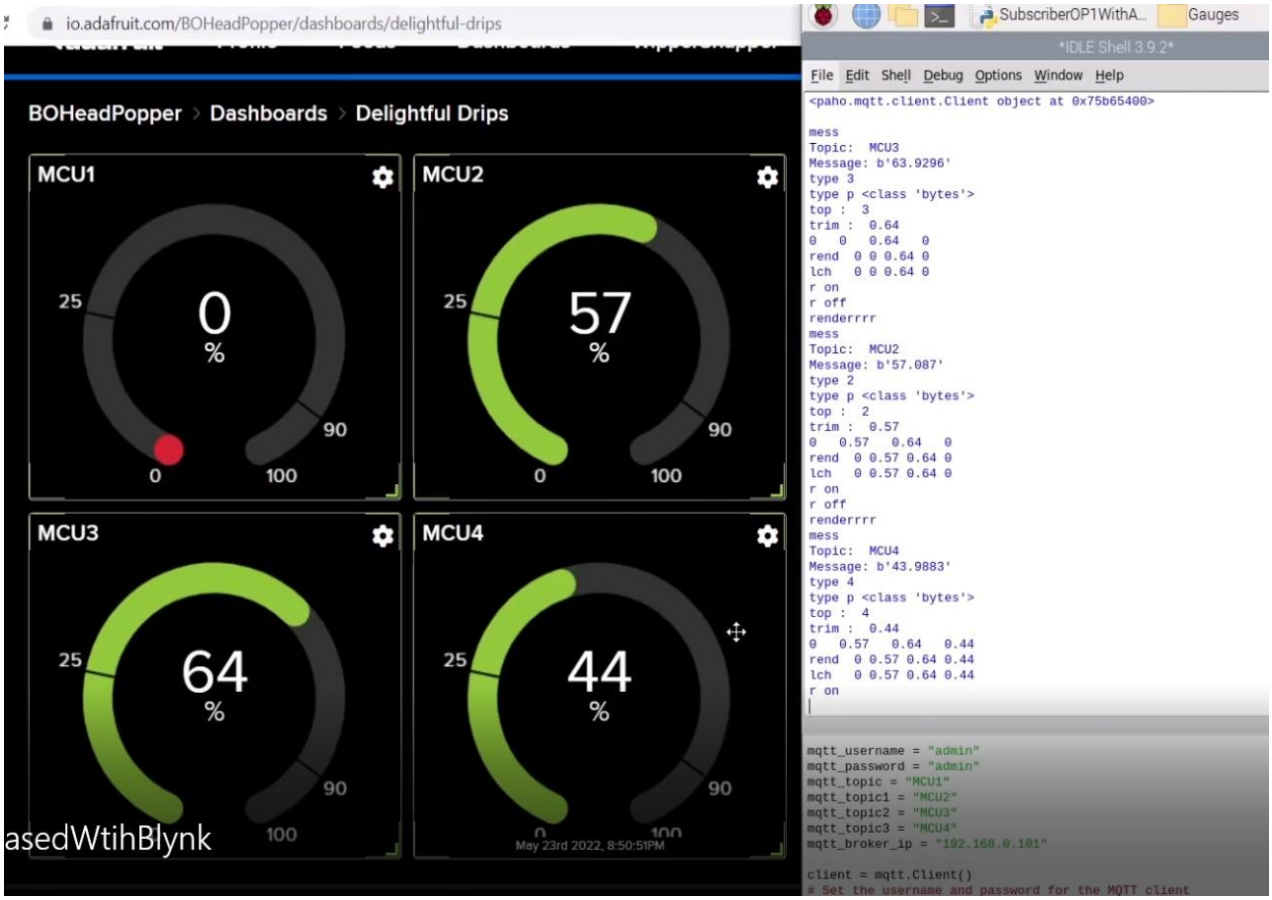


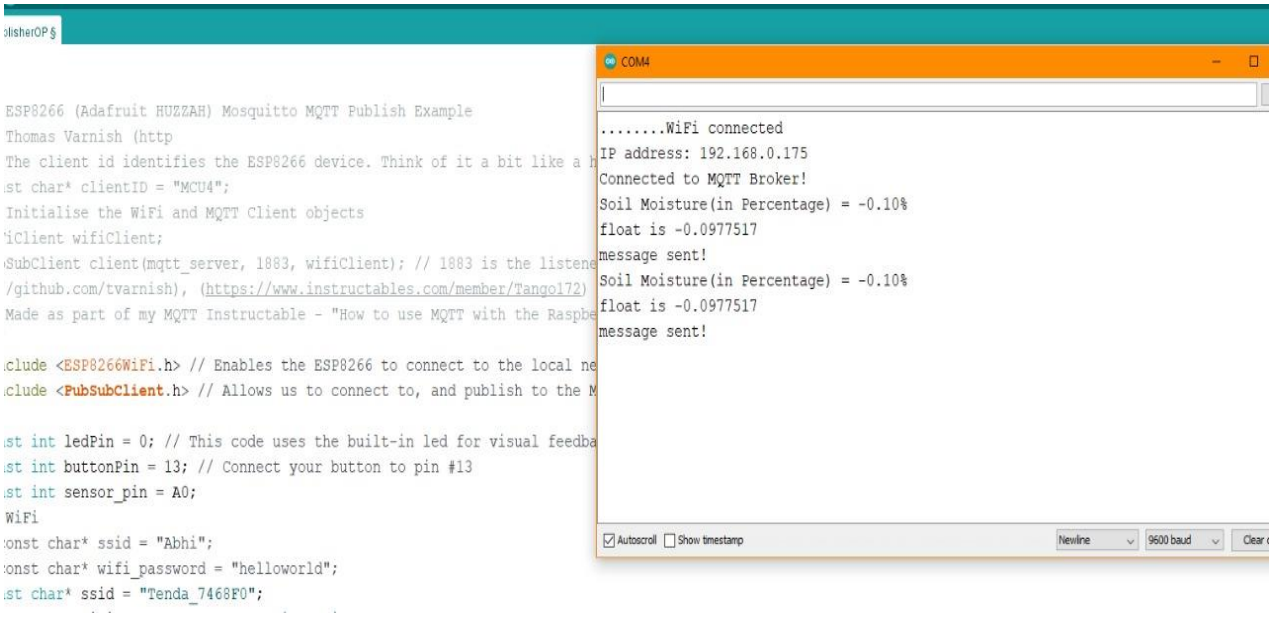
SubscriberOP1WithAdafruit.py - /home/pi/Desktop/SubscriberOP1WithAdafruit.py (3.9.2)

```
<paho.mqtt.client.Client object at 0x7497ca30>

mess
Topic:  MCU1
Message: b'0'
type 1
type p <class 'bytes'>
top :  1
trim :  0
0   0.63   0.63   0.45
rend  0 0.63 0.63 0.45
lch   0 0.63 0.63 0.45
r on
r off
renderrrr
mess
Topic:  MCU3
Message: b'64.4184'
type 3
type p <class 'bytes'>
top :  3
trim :  0.64
0   0.63   0.64   0.45
rend  0 0.63 0.64 0.45
lch   0 0.63 0.64 0.45
r on
r off
renderrrr
mess
Topic:  MCU2
Message: b'62.1701'
type 2
type p <class 'bytes'>
top :  2
trim :  0.62
0   0.62   0.64   0.45
rend  0 0.62 0.64 0.45
lch   0 0.62 0.64 0.45
r on
```

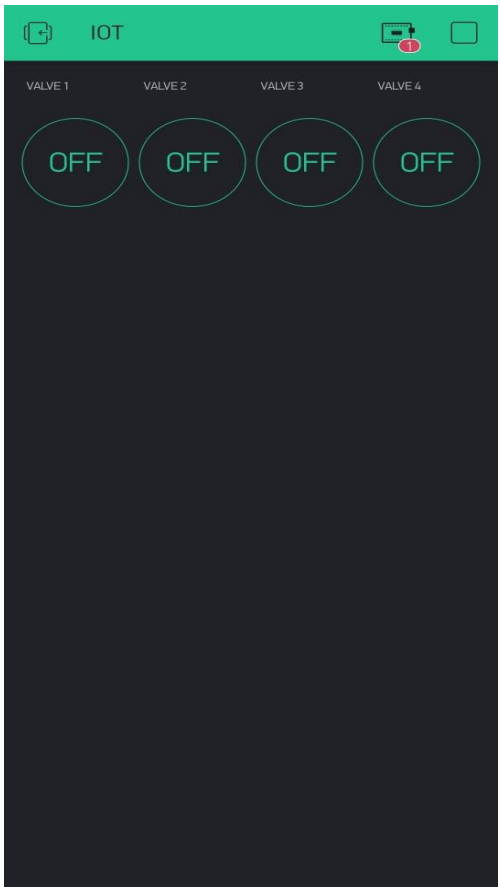
Ln: 6 Col: 0

```
mqtt_username = "admin"
mqtt_password = "admin"
mqtt_topic = "MCU1"
mqtt_topic1 = "MCU2"
mqtt_topic2 = "MCU3"
mqtt_topic3 = "MCU4"
mqtt_broker_ip = "192.168.0.101"

client = mqtt.Client()
# Set the username and password for the MQTT client
client.username_pw_set(mqtt_username, mqtt_password)
```


io.adafruit.com/BOHeadPopper/dashboards/delightful-drips
BOHeadPopper › Dashboards › Delightful Drips
MCU1
0
%
MCU2
57
%
MCU3
64
%
MCU4
44
%
May 23rd 2022, 8:50:51PM
SubscriberOP1WithA...
Gauges
IDLE Shell 3.9.2
File Edit Shell Debug Options Window Help
<paho.mqtt.client.Client object at 0x75b65400>
mess
Topic: MCU3
Message: b'63.9296'
type 3
type p <class 'bytes'>
top : 3
trim : 0.64
0 0 0.64 0
rend 0 0 0.64 0
lch 0 0 0.64 0
r on
r off
renderrrr
mess
Topic: MCU2
Message: b'57.087'
type 2
type p <class 'bytes'>
top : 2
trim : 0.57
0 0.57 0.64 0
rend 0 0.57 0.64 0
lch 0 0.57 0.64 0
r on
r off
renderrrr
mess
Topic: MCU4
Message: b'43.9883'
type 4
type p <class 'bytes'>
top : 4
trim : 0.44
0 0.57 0.64 0.44
rend 0 0.57 0.64 0.44
lch 0 0.57 0.64 0.44
r on
mqtt_username = "admin"
mqtt_password = "admin"
mqtt_topic = "MCU1"
mqtt_topic1 = "MCU2"
mqtt_topic2 = "MCU3"
mqtt_topic3 = "MCU4"
mqtt_broker_ip = "192.168.0.101"
client = mqtt.Client()
# Set the username and password for the MQTT client


ESP8266 (Adafruit HUZZAH) Mosquitto MQTT Publish Example
Thomas Varnish (http
The client id identifies the ESP8266 device. Think of it a bit like a h
st char* clientID = "MCU4";
Initialise the WiFi and MQTT Client objects
iClient wifiClient;
SubClient client(mqtt_server, 1883, wifiClient); // 1883 is the listene
/github.com/tvarnish), (https://www.instructables.com/member/Tango172)
Made as part of my MQTT Instructable - "How to use MQTT with the Raspbe
clude <ESP8266WiFi.h> // Enables the ESP8266 to connect to the local ne
clude <PubSubClient.h> // Allows us to connect to, and publish to the M
st int ledPin = 0; // This code uses the built-in led for visual feedba
st int buttonPin = 13; // Connect your button to pin #13
st int sensor_pin = A0;
WiFi
onst char* ssid = "Abhi";
onst char* wifi_password = "helloworld";
st char* ssid = "Tenda_7468F0";
COM4
.....WiFi connected
IP address: 192.168.0.175
Connected to MQTT Broker!
Soil Moisture(in Percentage) = -0.10%
float is -0.0977517
message sent!
Soil Moisture(in Percentage) = -0.10%
float is -0.0977517
message sent!
Autoscroll
Show timestamp
Newline
9600 baud


mess
Topic: MCU3
Message: b'62.4633'
type 3
type p <class 'bytes'>
top : 3
trim : 0.62
0.69 0.62 0.62 0.47
rend 0.69 0.62 0.62 0.47
lch 0.69 0.62 0.62 0.47
renderrrr
mess
Topic: MCU2
Message: b'61.8768'
type 2
type p <class 'bytes'>
top : 2
trim : 0.62
0.69 0.62 0.62 0.47
rend 0.69 0.62 0.62 0.47
lch 0.69 0.62 0.62 0.47
renderrrr
mess
Topic: MCU4
Message: b'45.9433'
type 4
type p <class 'bytes'>
top : 4
trim : 0.46
0.69 0.62 0.62 0.46
rend 0.69 0.62 0.62 0.46
lch 0.69 0.62 0.62 0.46
renderrrr
mess
Topic: MCU1
Message: b'69.306'
type 1
type p <class 'bytes'>
top : 1
trim : 0.69
Ln: 6 Col: 0
mqtt_username = "admin"
mqtt_password = "admin"
mqtt_topic = "MCU1"
mqtt_topic1 = "MCU2"
mqtt_topic2 = "MCU3"
hemeBasedWtihBlynk
client = mqtt.Client()
# Set the username and password for the MQTT client
Soil
69%
Gauge 1
62%
Gauge 2
62%
Gauge 3
46%
Gauge 4


IOT
VALVE 1
VALVE 2
VALVE 3
VALVE 4
OFF
OFF
OFF
OFF

6. CONCLUSION AND FUTURE WORK

The main aim was to develop a cost effective, smart irrigation system that operates based on the real-time moisture content in the soil.

We would like to scale this project in the future so that this project can work accurately for a larger piece of land. This can be done by installing more number of NodeMCUs in the field, so that the moisture content in the soil can be detected and sent to the Raspberry Pi for the automatic irrigation control.

7. REFERENCES

- https://www.instructables.com/How-to-Use-MQTT-With-the-Raspberry-Pi-and-ESP8266/
- http://esp8266.net/
- https://arduino-esp8266.readthedocs.io/en/latest/
- https://roboindia.com/tutorials/blynk-raspberry-digitaloutput/
- https://stackoverflow.com/
- https://www.instructables.com/Interface-Moisture-Sensor-With-NodeMCU/
- https://www.youtube.com/
- https://en.wikipedia.org/