Mini Project Report

on

CROP MONITORING SYSTEM USING ESP32 AND LOCAL SERVER

In partial fulfillment of the requirements for the award of the degree

BACHELOR OF TECHNOLOGY

in

Electronics and Communication Engineering

Submitted By

G. Sai Rohith Vishaal – 17R11A0462

D. Bhargavi – 18R15A0415

T. Raju – 18R15A0419

Under the Guidance of

Mrs. Padmaja A. R. L

Assistant Professor

Department of Electronics and Communication Engineering

GEETHANJALI COLLEGE OF ENGINEERING AND TECHNOLOGY,

(UGC AUTONOMOUS)

2020-2021

GEETHANJALI COLLEGE OF ENGINEERING AND TECHNOLOGY

Department of Electronics and Communication Engineering

<u>CERTIFICATE</u>

This is to certify that the project report titled CROP MONITORING SYSTEM USING ESP32 AND LOCAL SERVER being submitted by **G. Sai Rohith Vishaal, D. Bhargavi, T. Raju** bearing Registered Numbers **17R11A0462, 1815A0415, 18R15A0419** respectively**,** in partial fulfilment of the requirements for the award of the Degree of **Bachelor of Technology** in *Electronics and Communication Engineering* is a record of bona-fide work carried out under my guidance and supervision. The results embodied in this report have not been submitted to any other University for the award of any degree.

Internal Supervisor   **Dr. S. Suryanarayana**

Mrs. Padmaja A. R. L   **HOD, ECE**

Assistant Professor

ACKNOWLEDGEMENT

We, the Students of ECE department of Geethanjali College of Engineering and Technology would like to convey heartfelt thanks to **Dr. S. Udaya Kumar**, Principal of the college for the wonderful guidance and encouragement given to us to move ahead in the execution of this project.

We are highly grateful to the great personality in the field of Electronics, none other than **Dr. S. Suryanarayana**, Head of the Department of **Electronics and Communication Engineering** of **GCET** for guiding and taking care of our career in this field.

We are very happy for being guided by **Mrs. Padmaja A. R. L,** Assistant Professor for her able guidance which was given to us to complete our technical work successfully.

Above all, we are very much thankful to the **management of Geethanjali College of Engineering & Technology** which was established by the **high profile intellectuals** for the cause of Technical Education in modern era and produce uncountable young engineers and present them to the modern technical world.

With Regards,
G. Sai Rohith Vishaal – 17R11A0462
D. Bhargavi – 18R15A0415
T. Raju – 18R15A041

CONTENTS

LIST OF FIGURES

<u>ABSTRACT</u>

Aim of the Project:

- To design a crop monitoring system based on the Internet of things (IoT) and increase the productivity of the farmer in growing a particular crop in a Green House.

Problem Analysis:

- **Optimised Conditions:**
  - Primarily we need to know the optimized conditions for the plant (example: tomato), that are suitable to produce higher and effective yield.

- **Variables:**
  - Variables such as pH value, temperature, light levels, humidity, soil moisture should be monitored for ideal yield.

- **Data Relay:**
  - A local server is used to gather data from the sensors to perform analysis and extract the necessary statistics.

- **Application:**
  - The farmer or the user needs an application where all the data is organized and can be controlled anywhere in the world and it should help him/her to make the decisions.

Proposed Solution:

- **Brain of the Project:**
  - We are using ESP32 as the brain of the project to sense and actuate things.

- **Monitoring Variables:**
  - pH sensor, Digital Humidity and temperature sensor, Light Dependent Resistor, Soil Moisture sensor are used to measure pH levels, temperature, humidity, luminosity, soil moisture levels respectively.
  - Water pump, servo motors attached to windows, solenoid valve to control water distribution, light inlet and allowing water respectively.

- **Data Relay:**
  - For relaying data, we chose to use a server running mosquitto software that is hosted locally.
  - The mosquitto library works on MQTT protocol.
  - On the other end, user will have an app that displays everything the MCU is performing in an organized way and it even lets you control it.

CHAPTER 1 – INTRODUCTION

The world over decades has made considerable advancement in automation. Automation is employed for every sector where it is home, industry agriculture. Greenhouse is the technical approach in which farmers in the rural areas will be benefitted by automatic monitoring& control of greenhouse environment replace the direct supervisions of the human. The paper focuses on the generic architecture which can be applied for many other automation applications. The great needs are growing of crops with advancement of technology. Greenhouse are climate-controlled structure with wall &roofs &specially designed for off season growing of plants. Internet of things is one of the latest advances in information &communication technologies providing global connectivity &management of sensors devices, users with information.

Temperature/Humidity Sensor, Moisture Sensor, Light Sensor efficiently inside the greenhouse by actuating a Cooling Fan, LED, Motor respectively according to the required conditions of the crops to achieve the maximum growth &yield. First let us discuss about green house & greenhouse effect. Green house is something related to a building or a place where small plants & vegetables are grown. And the area under greenhouse is covered with glass or translucent plastic roof sand this plays & important role for the vegetation in cold regions, because it is still very cold to take them to an outside environment. And now moving forward to discuss about the greenhouse effect.

Greenhouse effect is simply a process in which various greenhouse gases entraps the infrared rays from the sunlight thus leading to increase of level of carbon dioxide which further helps in increasing the amount of chlorophyll and the leading to impressive plant growth & yield. And the greenhouse system helps in boosting the efficiency. And thus, our system is based to perform such activities that are to monitor& control the system from a particular place which would take care weather inside the greenhouse.

CHAPTER 2 – LITERATURE SURVEY

Automated greenhouse system helps the farmers by controlling the environment parameters through the environmental parameters through the internet of things (IOT) including crop health inspection using image analysis the greenhouse is generally affected by two factors: plant diseases &weather condition, which leads to the fall in production.

The weather condition can be controlled through Microcontroller Unit (MCU) & the plant diseases can be monitored using image inspection system. The research recommends cheaper image evaluation framework for the plant disease can be monitored using image inspection system. The research recommends cheaper image evaluation framework for the plant diseases analysis& fully automated greenhouse data security. The prototype of the proposed system consists of Temperature/Humidity Sensor, Moisture Sensor, Light Sensor and pH Sensor.

The Motor, Light, are controlled by ESP32 through an app upon reaching predetermined threshold values. The proposed architecture is equipped with embedded data security by implementing Extended Tiny Encryption Algorithms (XTEA) lastly, the agriculturists can familiarize with the recommended framework through the cloud-centric application. The autonomous frameworks permit the agriculturists to evaluate &control their greenhouse ecology remotely.

 (Majone, F. Viani, E. Filippi, A. Bellin, A. Massa, G. Toller, F. Robol and M. Salucci "Wireless Sensor Network deployment for monitoring soil moisture dynamics at the field scale", Elsevier, 2013, vol.19, pp. 426- 435.)

CHAPTER 3 – DESIGN
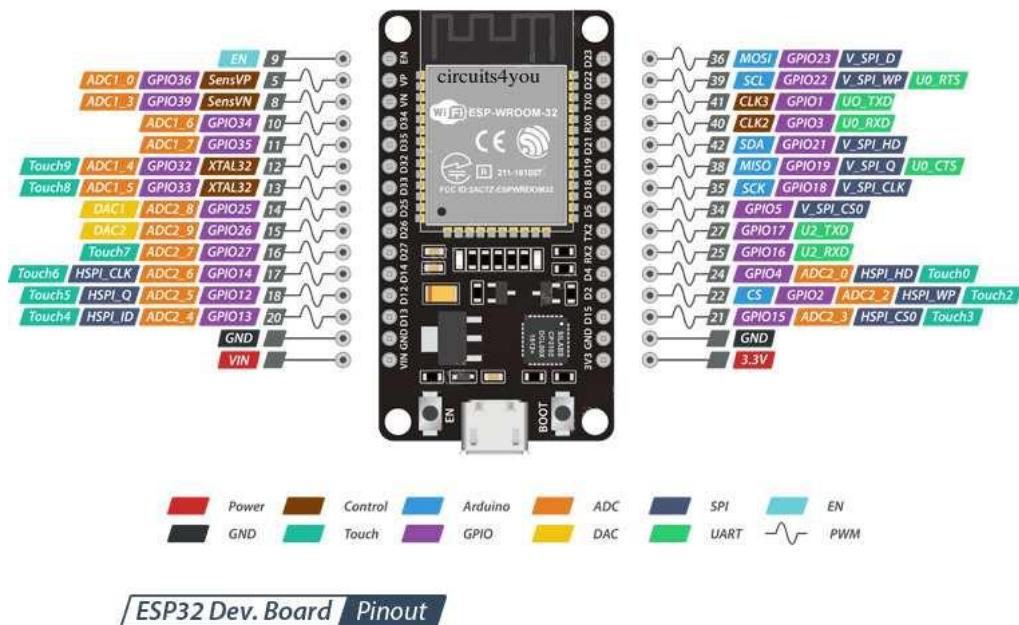
3.1 – COMPONENTS DESCRIPTION

Figure i - Chapter 3.1 - ESP32 Pinout

ESP32-WROOM-32 is a powerful, generic Wi-Fi+BT+BLE MCU module that targets a wide variety of applications, ranging from low-power sensor networks to the most demanding tasks, such as voice encoding, music streaming and MP3 decoding. At the core of this module is the ESP32-D0WDQ6 chip*. The chip embedded is designed to be scalable and adaptive. There are two CPU cores that can be individually controlled, and the CPU clock frequency is adjustable from 80 MHz to 240 MHz The user may also power off the CPU and make use of the low-power co-processor to constantly monitor the peripherals for changes or crossing of thresholds. ESP32 integrates a rich set of peripherals, ranging from capacitive touch sensors, Hall sensors, SD card interface, Ethernet, high-speed SPI, UART, I2S and I2C.

The integration of Bluetooth, Bluetooth LE and Wi-Fi ensures that a wide range of applications can be targeted, and that the module is future proof: using Wi-Fi allows a large physical range and direct connection to the internet through a Wi-Fi router, while using Bluetooth allows the user to conveniently connect to the phone or broadcast low energy beacons for its detection.

ESP32 Peripherals Features

- 18 Analog-to-Digital Converter (ADC) channels
- 10 Capacitive sensing GPIOs
- 3 UART interfaces
- 3 SPI interfaces
- 2 I2C interfaces

- 16 PWM output channels

- 2 Digital-to-Analog Converters (DAC)
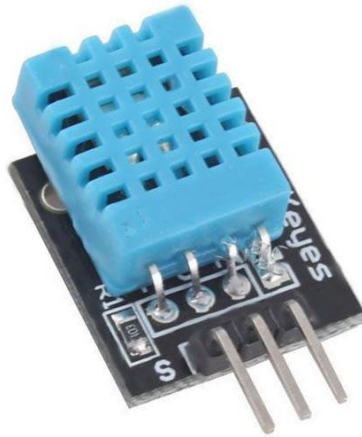
- 2 I2S interfaces

Figure ii - Chapter 3.1 - DHT 11

Working Principle of DHT11:

Temperature:

The temperature is measured with the help of a NTC thermistor or negative temperature coefficient thermistor. These thermistors are usually made with semiconductors, ceramic and polymers. The resistance of the device is inversely proportional with temperature and follows a hyperbolic curve. Temperature using NTC often found out Steinhart Hart equation.

Humidity:

Humidity is sensed using a moisture dependent resistor. It has two electrodes and in between them there exist a moisture holding substrate which holds moisture. The Conductance and hence resistance changes with changing humidity.

DHT11 Specifications:

- Operating Voltage: 3.5V to 5.5V

- Operating current: 0.3mA (measuring) 60uA (standby)

- Output: Serial data

- Temperature Range: 0°C to 50°C

- Humidity Range: 20% to 90%

- Resolution: Temperature and Humidity both are 16-bit
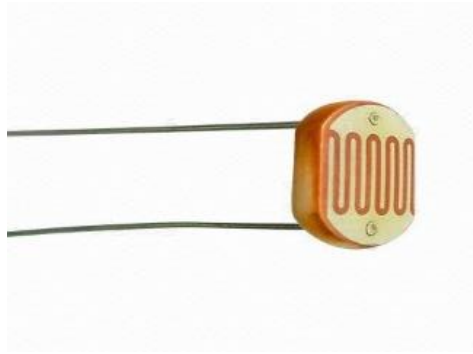
- Accuracy: ±1°C and ±1%

Figure iii - Chapter 3.1 - Light Dependent Resistor(LDR)

Specifications:

- Maximum power dissipation   200Mw
- Maximum voltage @ 0lux   200V
- Peak wavelength   600nm
- Minimum resistance @ 10 lux   1.8Ω
- Maximum resistance @10 lux   4.5KΩ
- Dark resistance after 1 second   0.03MΩ
- Dark resistance after 5 seconds   0.25MΩ

Working Principle of LDR:

- A light dependent resistor works on the principle of photo conductivity. Photo conductivity is an optical phenomenon.

- When light falls i.e. when the photons fall on the device, the electrons in the valence band of the semiconductor material are excited to the conduction band. These photons in the incident light should have energy greater than the band gap of the semiconductor material to make the electrons jump from the valence band to the conduction band.

- Hence when light having enough energy strikes on the device, more and more electrons are excited to the conduction band which results in large number of charge carriers. The result of this process is more and more current starts flowing through the device when the circuit is closed and hence it is said that the resistance of the device has been decrease.

Figure iv - Chapter 3.1 - Soil Moisture Sensor

Specifications:

- Range: 0 to 45% volumetric water content in soil (capable of 0 to 100% VWC with alternate calibration)
- Accuracy: ±4% typical
- Typical Resolution:0.1%
- Power: 3mA@ 5VDC
- Operating temperature: -40℃ to +60℃.

  The Soil Moisture Sensor uses capacitance to measure the water content of soil (by measuring the dielectric permittivity of the soil, which is a function of the water content). Simply insert this rugged sensor into the soil to be tested, and the volumetric water content of the soil is reported in percent.

Working Principle of Moisture Sensor:

- The Soil Moisture Sensor uses capacitance to measure dielectric permittivity of the surrounding medium. In soil, dielectric permittivity is a function of the water content. The sensor creates a voltage proportional to the dielectric permittivity, and therefore the water content of the soil.

- The Soil Moisture Sensor is used to measure the loss of moisture over time due to evaporation and plant, evaluate optimum soil moisture contents for various species of plants, monitor soil moisture content to control irrigation in greenhouses and enhance yield of the plant.

Figure v - Chapter 3.1 - pH Sensor

Specifications:

- Type: Sealed, gel-filled, epoxy body, Ag/AgCl.

- Shaft Diameter: 12mm OD.

- Response time: 90% of final reading in 1 second.

- Temperature Range:5 to 80°C.

- Range: pH 0-14.

- Accuracy: ±0.2 pH units.

- ISO-potential pH: pH 7(point at which temperature has no effect).

Soil content and quality is a key factor of crop health. Physically, soil material and density affects water infiltration and retention. Chemically, pH of soil affects nutrient availability and uptake. Some soil characteristics are permanent qualities of a soil, and therefore part of soil selection criteria, others s by measuring soil using a soil pH meter.

pH monitoring is a crucial component of horticulture maintenance for crop health. The best pH for plants is typically can be controlled and optimized. The best way to start the process of improving soil health between 5.5 and 6.5, though some plants may thrive in more acidic or more alkaline soils. Soil nutrients are tied strongly with pH of soil, pH control maximizes the efficiency of fertilizers by controlling nutrient bioavailability pH of soils affects the presence of toxic elements structure of certain soils, and the activity of soil bacteria.

Figure vi - Chapter 3.1 - Servo Motor

Specifications:

- Operating Voltage: +5V typically

- Torque: 2.5kg/cm

- Operating speed: 0.1s/60◦

- Gear Type: Plastic

- Rotation: 0◦-180◦

- Weight of motor: 9gm

- Package includes gear horns and screws

Servo motor is a tiny and light weight server motor with high output power. Servos are controlled by sending an electrical pulse of variable width or pulse width modulation (PWM), through the control wire. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but smaller. Each part of the servo motor like stator, winding, shaft, rotor, encoder makes   the servo properly function. The motor with power +5V using the red wire and brown wire is ground wire connected to the ground of system, orange wire is PWM signal is given in through this wire to drive the motor.

Figure vii - Chapter 3.1 - Water Pump

Specifications:

- Operating Voltage: 2.5V-3V

- Operating Current: 130 -220mA

- Flow Rate: 80 -120 L/H

- Maximum Lift: 40 -110mm

- Continuous Working Life: 500 hours

- Driving Mode: DC, Magnetic Driving

- Material: Engineering Plastic

This is micro submersible water pump dc 3v-5v, can be easily integrate to water system. This water pump works using water suction method which drain the water through its inlet and released it through the outlet. To distribute water to the crops by drip irrigation method.

Figure viii - Chapter 3.1 - Thermostat

Specifications:

- Screen: 24-bit colour LCD

  320 * 320pixel Display

  1.75 in (44.5mm) diameter

- Battery: Built -in rechargeable lithium-ion battery

- Power: Less than 1KWh/month

- Connectivity requirement: Wi-Fi internet connection

- Operating conditions: Temperature-32°F-104°F (0°C-40°C)

   Humidity- Up to 90% RH unpackaged Pressure -Up to 10,000
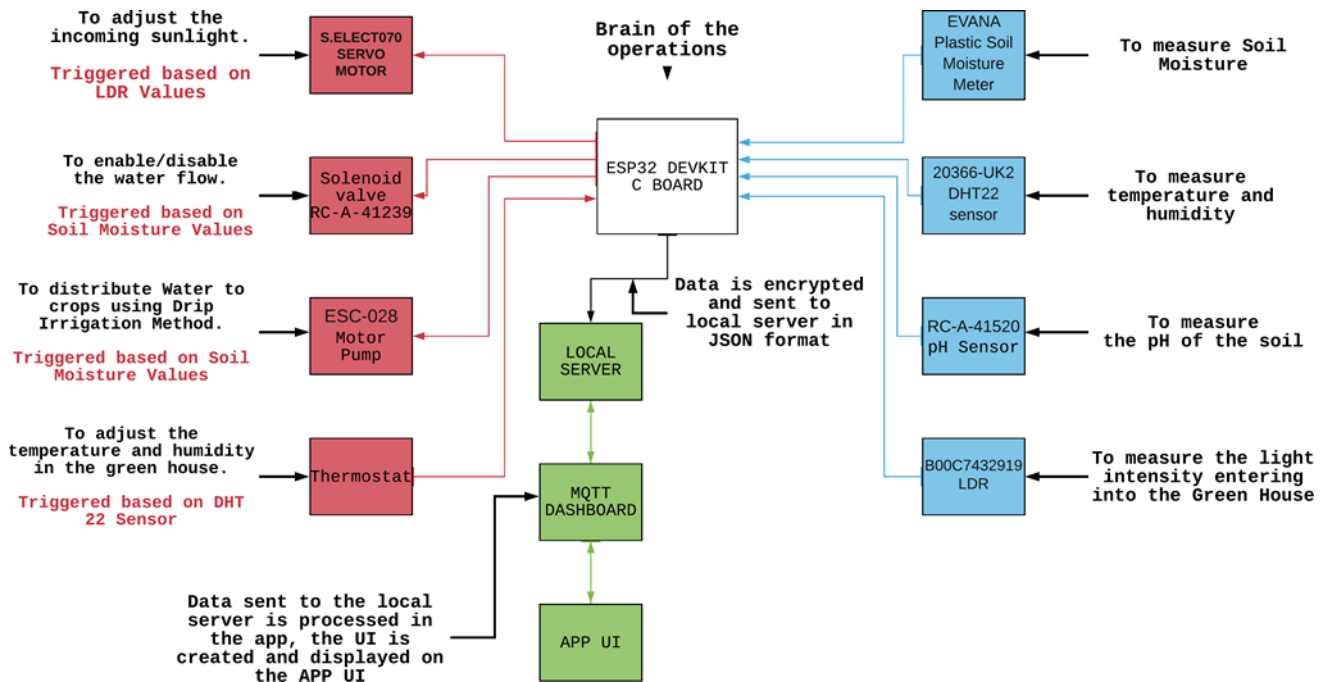
3.2 – BLOCK DIAGRAM

Figure ix - Chapter 3.1 - Block Diagram

- The ESP32 micro controller will read values from the sensors and store them in its memory.
- The user can define threshold values for each sensor in the app.
- The data from the app is relayed to ESP32 via the mosquitto broker that is hosted on the local server.
- Now, the ESP32 will continue to monitor over the thresholds. If the threshold is crossed a coded action will be executed.
- Suppose if the LDR's higher threshold limit is crossed. We can tell to the micro controller to close the windows, so that the amount of light intercepting on the plant is decreased.
- In a similar fashion, actions required to neutralise the situations need to be performed.
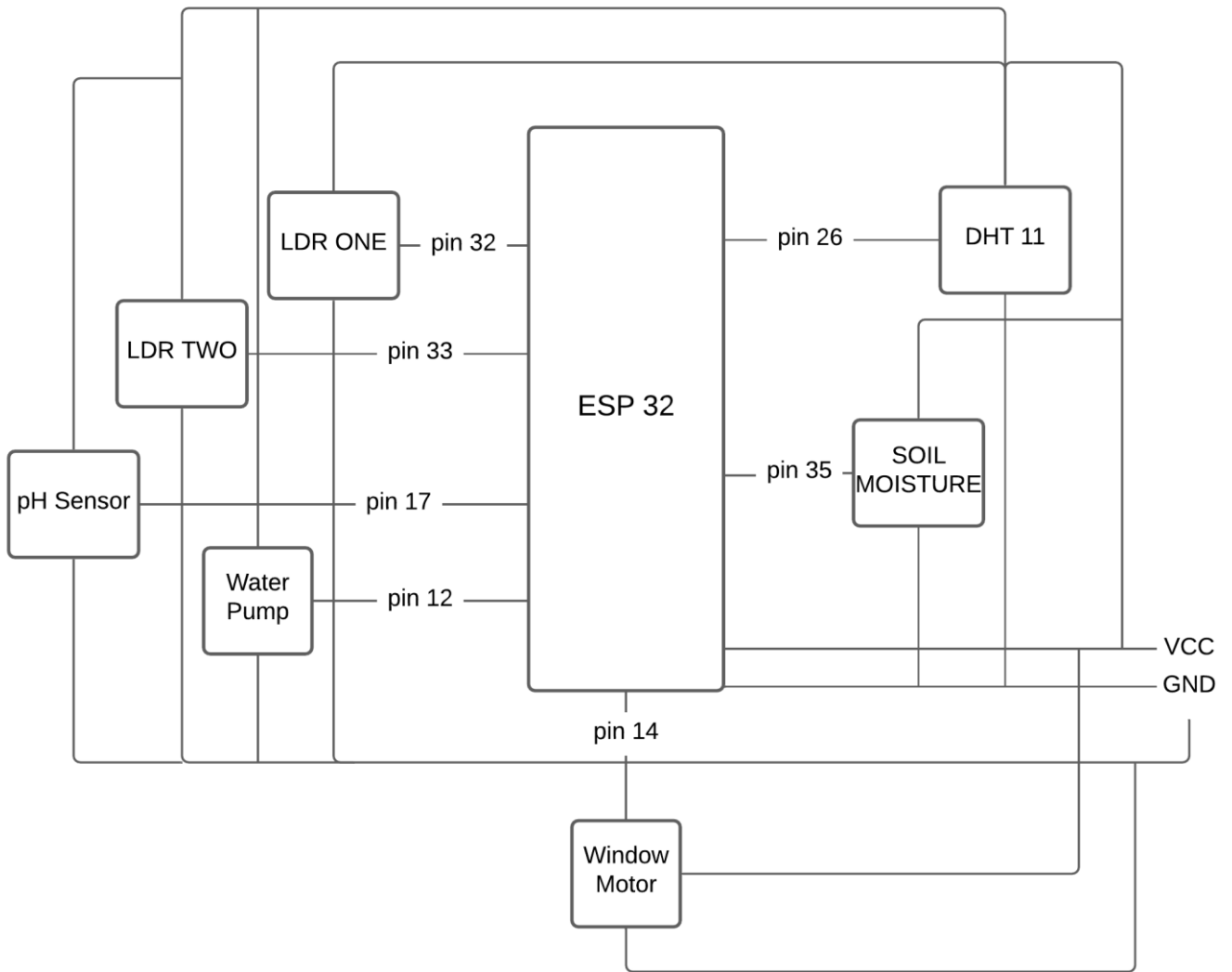- The rest of the block diagram is self-explanatory.

3.3 – CIRCUIT DIAGRAM

Figure x - Chapter 3.2 - Circuit Diagram
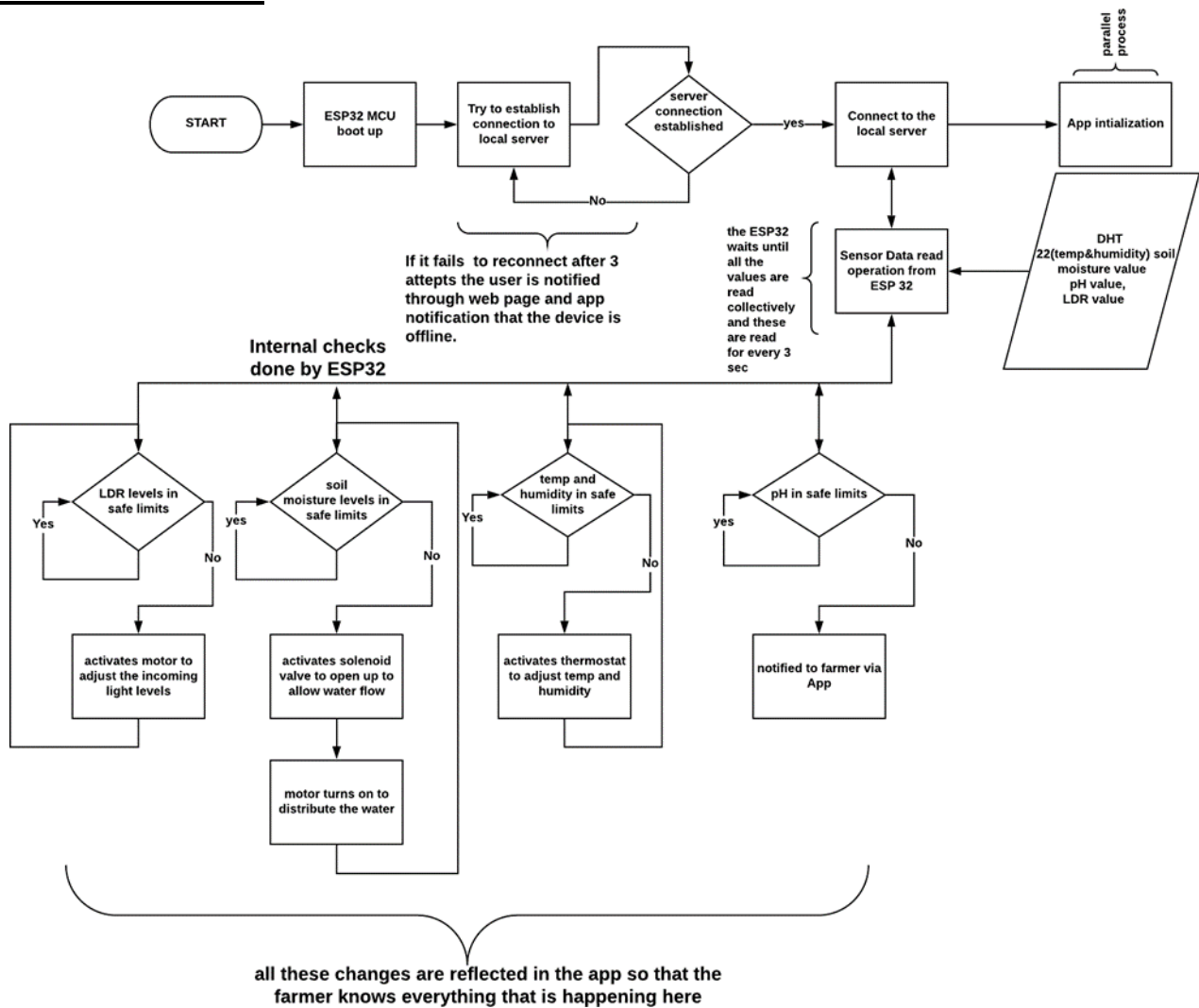
3.4 – FLOW CHART

Figure xi - Chapter 3.3 - Flow Chart

- Initially ESP32 will boot up, then it will try to establish internet connection and later it will connect to the MQTT broker.
- If the process is failed, it will re-try 3 times to establish a connection. If the connection is unsuccessful the same is notified in the app because the connection is mutual and important to both the board and the app.
- If the connection is successful, the app will initialize with the previous values set and the same values are sent to ESP32 for setting its internal threshold values for each sensor.
- Once everything is set ESP32 will continue to monitor values from sensor and they are compared against the threshold values.
- These threshold values are set by the user from the app or some default settings are set in the code itself.
- As you can see from the above flow chart diagram, if the thresholds are crossed a particular action is performed in-order to neutralize the effect.

3.5 – PROGRAM CODE

```
// Necessary Libraries
#include <AsyncMqttClient.h>
#include <WiFi.h>
extern "C" {
#include "freertos/FreeRTOS.h"
#include "freertos/timers.h"
}


// For Temperature and Humidity
#include <DHT.h>


// For Servo that is the window motor
#include <ESP32Servo.h>


// WiFi configuration
#define WIFI_SSID  "<< your wifi ssid >>"
#define WIFI_PASSWORD  "<< your wifi password >> "


// MQTT configuration
// make sure it is a static IP
// If the IP is not static set it up as a static IP in the router configuration
// If you using a cloud MQTT broker type in the domain name
// #define MQTT_HOST "io.adafruit.com"
#define MQTT_HOST "192.168.31.169"
// This port is insecure (not encrypted) but if your running it locally it's okay if your wifi is not hacked
// If you are using a cloud broker make sure you use an SSL connection.
// https://www.digitalocean.com/community/tutorials/how-to-install-and-secure-the-mosquitto-mqtt-
messaging-broker-on-ubuntu-16-04
#define MQTT_PORT 1883
// Setting up authentication for mosquitto broker
```

```
// https://medium.com/@eranda/setting-up-authentication-on-mosquitto-mqtt-broker-de5df2e29afc
#define MQTT_USERNAME "<< your username >>"
#define MQTT_PASS "<< your password >>"


// MQTT Topics
// Sensor topics
#define MQTT_PUB_TEMP           "greenhouse/feeds/temperature"
#define MQTT_PUB_HUM            "greenhouse/feeds/humidity"
#define MQTT_PUB_LDR1           "greenhouse/feeds/ldr1"
#define MQTT_PUB_LDR2           "greenhouse/feeds/ldr2"
#define MQTT_PUB_SOIL_MOISTURE     "greenhouse/feeds/soil-moisture"
#define MQTT_PUB_PH_LEVEL          "greenhouse/feeds/pH-level"


// Threshold Settings topics
#define MQTT_SUB_TEMP_LOW_SET          "greenhouse/feeds/temperature-low-set"
#define MQTT_SUB_TEMP_HIGH_SET         "greenhouse/feeds/temperature-high-set"
#define MQTT_SUB_HUM_LOW_SET           "greenhouse/feeds/humidity-low-set"
#define MQTT_SUB_HUM_HIGH_SET          "greenhouse/feeds/humidity-high-set"
#define MQTT_SUB_LDR1_LOW_SET          "greenhouse/feeds/ldr1-low-set"
#define MQTT_SUB_LDR1_HIGH_SET         "greenhouse/feeds/ldr1-high-set"
#define MQTT_SUB_LDR2_LOW_SET          "greenhouse/feeds/ldr2-low-set"
#define MQTT_SUB_LDR2_HIGH_SET         "greenhouse/feeds/ldr2-high-set"
#define MQTT_SUB_PH_LEVEL_LOW_SET      "greenhouse/feeds/pH-level-low-set"
#define MQTT_SUB_PH_LEVEL_HIGH_SET     "greenhouse/feeds/pH-level-high-set"
#define MQTT_SUB_SOIL_MOISTURE_LOW_SET  "greenhouse/feeds/soil-moisture-low-set"
#define MQTT_SUB_SOIL_MOISTURE_HIGH_SET "greenhouse/feeds/soil-moisture-high-set"


// User Actions topics
#define MQTT_SUB_WATER_PUMP_SWITCH "greenhouse/feeds/water-pump-switch"
#define MQTT_SUB_WINDOW_SWITCH     "greenhouse/feeds/window-switch"
#define MQTT_SUB_MANUAL_OVERRIDE   "greenhouse/feeds/manual-override"
```

```
// Feedback topics
#define MQTT_PUB_STATUS        "greenhouse/feeds/status"
#define MQTT_PUB_STATUS1       "greenhouse/feeds/status1"


// Board : https://circuits4you.com/2018/12/31/esp32-devkit-esp32-wroom-gpio-pinout/
// Pin Configuration for Sensors
#define DHTPIN      26
#define SOILMOISTURE 35
#define PH          17
#define LDR1        32
#define LDR2        33


// Pin Configuration for actuators
#define WATERPUMP    12
#define WINDOW_MOTOR 14


#define DHTTYPE DHT11
// create a dht object
DHT dht(DHTPIN, DHTTYPE);


// Initializing a MQTT client Object
AsyncMqttClient mqttClient;


// Creates a new software timer instance and returns a handle by which the timer can be referenced.
// Doc:https://www.freertos.org/FreeRTOS-timers-xTimerCreate.html
TimerHandle_t mqttReconnectTimer;
TimerHandle_t wifiReconnectTimer;


// Stores last time when data was published
unsigned long previousMillis = 0;
unsigned long previousSubMillis = 0;
// Interval at which to publish the sensor readings
const long interval = 5000;
```

```
const long subscribeInterval = 2000;
//  Variables to hold sensor readings
float temperature = 26;
float humidity = 62;
int soilMoisture = 55;
int ldrOne = 62, ldrTwo = 75;
float pH = 6;


// Variables to hold theresholds from the app
int tempLowThreshold = 25, tempHighThreshold = 40;
int humLowThreshold = 30, humHighThreshold = 70;
int soilMoistureLowThreshold = 50, soilMoistureHighThreshold = 65;
int ldrOneLowThreshold = 20, ldrOneHighThreshold = 95;
int ldrTwoLowThreshold = 20, ldrTwoHighThreshold = 95;
int pHLowThreshold = 5, pHHighThreshold = 12;


// setting ideal safe values
int tempSafe = (tempLowThreshold + tempHighThreshold)/2;
int humSafe = (humLowThreshold + humLowThreshold)/2;
int soilSafe = ( soilMoistureLowThreshold + soilMoistureHighThreshold)/2;
int ldrOneSafe =  (ldrOneLowThreshold + ldrOneHighThreshold)/2;
int ldrTwoSafe =  (ldrOneLowThreshold + ldrOneHighThreshold)/2;
int pHSafe = (pHLowThreshold+pHHighThreshold)/2;
boolean soilSafeState = false;


//User Actions
boolean manualOverride = false;


// first status
boolean initialStatus = true;
```

```cpp
// Servo object for the window switch
Servo windowMotor;

void connectToWifi() {
  Serial.println("[ACTION] -- Connecting to Wi-Fi...");
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}

void connectToMqtt() {
  Serial.println("[ACTION] -- Connecting to MQTT...");
  mqttClient.connect();
}

void WiFiEvent(WiFiEvent_t event) {
  switch (event) {
    case SYSTEM_EVENT_STA_GOT_IP:
      Serial.println("[STATUS] -- WiFi connected");
      Serial.println("[INFO] -- IP address: ");
      Serial.println(WiFi.localIP());
      connectToMqtt();
      break;

    case SYSTEM_EVENT_STA_DISCONNECTED:
      Serial.println("[ALERT] -- WiFi connection lost!");
      // Ensure we don't connect to MQTT broker when there is no WiFi
      xTimerStop(mqttReconnectTimer, 0);
      xTimerStart(wifiReconnectTimer, 0);
      break;
  }
}

void onMqttConnect(bool sessionPresent) {
```

```
Serial.println("[STATUS] -- Connected to MQTT.");
  Serial.print("[STATUS] -- Session Present:");
  Serial.println(sessionPresent);


  // When we connect to the MQTT we subscribe to topics
  uint16_t packetIdSub1 = mqttClient.subscribe(MQTT_SUB_TEMP_LOW_SET, 1);
  uint16_t packetIdSub2 = mqttClient.subscribe(MQTT_SUB_TEMP_HIGH_SET, 1);
  uint16_t packetIdSub3 = mqttClient.subscribe(MQTT_SUB_HUM_LOW_SET, 1);
  uint16_t packetIdSub4 = mqttClient.subscribe(MQTT_SUB_HUM_HIGH_SET, 1);
  uint16_t packetIdSub5 = mqttClient.subscribe(MQTT_SUB_SOIL_MOISTURE_LOW_SET, 1);
  uint16_t packetIdSub6 = mqttClient.subscribe(MQTT_SUB_SOIL_MOISTURE_HIGH_SET, 1);
  uint16_t packetIdSub7 = mqttClient.subscribe(MQTT_SUB_LDR1_LOW_SET, 1);
  uint16_t packetIdSub8 = mqttClient.subscribe(MQTT_SUB_LDR1_HIGH_SET, 1);
  uint16_t packetIdSub9 = mqttClient.subscribe(MQTT_SUB_LDR2_LOW_SET, 1);
  uint16_t packetIdSub10 = mqttClient.subscribe(MQTT_SUB_LDR2_HIGH_SET, 1);
  uint16_t packetIdSub11 = mqttClient.subscribe(MQTT_SUB_PH_LEVEL_LOW_SET, 1);
  uint16_t packetIdSub12 = mqttClient.subscribe(MQTT_SUB_PH_LEVEL_HIGH_SET, 1);
  uint16_t packetIdSub13 = mqttClient.subscribe(MQTT_SUB_MANUAL_OVERRIDE, 1);
 uint16_t packetIdSub14 = mqttClient.subscribe(MQTT_SUB_WATER_PUMP_SWITCH, 1);
  uint16_t packetIdSub15 = mqttClient.subscribe(MQTT_SUB_WINDOW_SWITCH, 1);
}


void onMqttDisconnect(AsyncMqttClientDisconnectReason reason) {
  Serial.println("[ALERT] -- Disconnected from MQTT.");
  if (WiFi.isConnected()) {
   // If the connection is lost to the MQTT.We check if the WiFi connection is present or not
   // Then we re-attempt to connect to the MQTT
   xTimerStart(mqttReconnectTimer, 0);
  }
}
```

```
void onMqttSubscribe(uint16_t packetId, uint8_t qos) {
  Serial.println("[STATUS] -- Subscribe acknowledged.");
  Serial.print("[INFO] -- packet ID:");
  Serial.print(packetId);
  Serial.print("[INFO] QOS: ");
  Serial.println(qos);
}


void onMqttPublish(uint16_t packetId) {
  Serial.printf("[INFO] -- Data Published with Packet ID:%d\n", packetId);
}


String valueExtract(char* payload, size_t len) {
  String temp = "";
  for (int i = 0; i <= len; i++) {
    temp += String((char)payload[i]);
  }
  Serial.println(temp);
 return temp;
}


void sendStatusFeedback(String msg, int channel) {
  if (channel == 1)
    mqttClient.publish(MQTT_PUB_STATUS, 1, true, String(msg).c_str());
  if (channel == 2)
    mqttClient.publish(MQTT_PUB_STATUS1, 1, true, String(msg).c_str());
}


void onMqttMessage(char* topic, char* payload, AsyncMqttClientMessageProperties properties, size_t len,
size_t index, size_t total) {
  Serial.println("[STATUS] -- Publish received.");
  Serial.print("[INFO] -- topic: ");
  Serial.println(topic);
```

```
 String value = valueExtract(payload, len);
  Serial.print(String(topic) + ":" + String(value));
  if (String(topic) == MQTT_SUB_TEMP_LOW_SET)
   tempLowThreshold = value.toInt()
  if (String(topic) == MQTT_SUB_TEMP_HIGH_SET)
   tempHighThreshold = value.toInt();
  if (String(topic) == MQTT_SUB_HUM_LOW_SET)
   humLowThreshold = value.toInt();
  if (String(topic) == MQTT_SUB_HUM_HIGH_SET)
   humHighThreshold = value.toInt();
  if (String(topic) == MQTT_SUB_SOIL_MOISTURE_LOW_SET)
   soilMoistureLowThreshold = value.toInt();
 if (String(topic) == MQTT_SUB_SOIL_MOISTURE_HIGH_SET)
   soilMoistureHighThreshold = value.toInt();
  if (String(topic) == MQTT_SUB_LDR1_LOW_SET)
   ldrOneLowThreshold = value.toInt();
  if (String(topic) == MQTT_SUB_LDR1_HIGH_SET)
   ldrOneHighThreshold = value.toInt();
  if (String(topic) == MQTT_SUB_LDR2_LOW_SET)
   ldrTwoLowThreshold = value.toInt();
  if (String(topic) == MQTT_SUB_LDR2_HIGH_SET)
   ldrTwoHighThreshold = value.toInt();
  if (String(topic) == MQTT_SUB_PH_LEVEL_LOW_SET)
   pHLowThreshold = value.toInt();
  if (String(topic) == MQTT_SUB_PH_LEVEL_HIGH_SET)
   pHHighThreshold = value.toInt();
  if (String(topic) == MQTT_SUB_MANUAL_OVERRIDE) {
   if (value[0] == 't') {
    manualOverride = true;
    Serial.printf("[INFO] -- Manual Ovveride ON");
    sendStatusFeedback("Manual Override ON!", 1);
   }
```

```
else {
    manualOverride = false;
    Serial.printf("[INFO] -- Manual Ovveride OFF");
    sendStatusFeedback("Manual Override OFF!", 1);
  }
 }

 if (String(topic) == MQTT_SUB_WATER_PUMP_SWITCH) {
  if (value[0] == 't' && manualOverride) {
    Serial.printf("[ACTION] -- MOTOR ON");
    sendStatusFeedback("MOTOR ON", 2);
    waterPump(true);
   }
   else if (value[0] == 'f' && manualOverride) {
    Serial.printf("[ACTION] -- MOTOR ON");
    sendStatusFeedback("MOTOR OFF", 2);
    waterPump(false);
   }
   else {
    Serial.printf("[ALERT] -- Cannot execute action! Manual Ovveride is OFF");
    sendStatusFeedback("Cannot execute action! Manual Override OFF!", 1);
   }
 }
 if (String(topic) == MQTT_SUB_WINDOW_SWITCH) {
  if (value[0] == 't' && manualOverride) {
    sendStatusFeedback("Opening windows", 2);
    Serial.printf("[ACTION] -- OPENING WINDOWS");
    windows(true);
   }
   else if (value[0] == 'f' && manualOverride) {
    sendStatusFeedback("Closing windows", 2);
     Serial.printf("[ACTION] -- CLOSING WINDOWS");
    windows(false);
```

```
  }
  else {
Serial.printf("[ALERT] -- Cannot execute action! Manual Ovveride is OFF");
    sendStatusFeedback("Cannot execute action! Manual Override OFF!", 1);
  }
 }
}
void setSafeValues(){
  //ideal mid values
tempSafe = (tempLowThreshold + tempHighThreshold)/2;
humSafe = (humLowThreshold + humLowThreshold)/2;
soilSafe = ( soilMoistureLowThreshold + soilMoistureHighThreshold)/2;
ldrOneSafe =  (ldrOneLowThreshold + ldrOneHighThreshold)/2;
ldrTwoSafe =  (ldrOneLowThreshold + ldrOneHighThreshold)/2;
pHSafe = (pHLowThreshold+pHHighThreshold)/2;
}


void getSensorData(boolean randomMode) {
 if (randomMode) {
   temperature = random(25, 31);
   humidity = random(60, 96);
   pH = random(5, 15);
   ldrOne = random(65, 100);
   ldrTwo = random(65, 100);
   soilMoisture = random(60, 96);
 }
 else {
   delay(500);
   temperature = dht.readTemperature();
   delay(500);
   humidity = dht.readHumidity();
   delay(500);
   if (isnan(temperature) || isnan(humidity))
```

Dept. of ECE, GCET

```
    Serial.println(F("[ALERT] -- Failed to read from DHT sensor!"));

    int ldrOneRead = analogRead(LDR1);

    delay(500);

    ldrOne = map(ldrOneRead, 0, 4095, 1, 100);

    delay(500);

    int ldrTwoRead = analogRead(LDR2);

    delay(500);

    ldrTwo = map(ldrTwoRead, 0, 4095, 1, 100);

    delay(500);

    int soilRead = analogRead(SOILMOISTURE);

    soilMoisture = 100 - map(soilRead, 0, 4095, 1, 100);

    delay(500);

    pH = random(4, 15);

  }

}


void thresholdChecks() {
  if (initialStatus) {
    sendStatusFeedback("ALL OK!", 2);
    initialStatus = false;
  }
  if (temperature < tempLowThreshold) {
    Serial.println("[ALERT] -- Temperature below Lower Threshold!");
    sendStatusFeedback("[ALERT] -- Temperature below Lower Threshold", 1);
    if (!manualOverride) {
      thermostat(true, tempLowThreshold + 2);
    }
  }
  if (temperature > tempHighThreshold) {
    Serial.println("[ALERT] -- Temperature above Higher Threshold!");
    sendStatusFeedback("Temperature above Higher Threshold", 1);
    if (!manualOverride) {
```

```
   waterPump(true);

   }

 }

 if(temperature > tempSafe -5 && temperature < tempSafe+5){

   Serial.println("[INFO] -- Temperature in safe levels!");

   sendStatusFeedback("Temperature in safe levels!", 1);

 }

 delay(500);

 if (humidity < humLowThreshold ) {

   Serial.println("[ALERT] -- Humidity below Lower threshold!");

   sendStatusFeedback("Humidity below Lower threshold!", 1);

   if (!manualOverride) {

     waterPump(true);

   }

 }

 if (humidity > humHighThreshold ) {

   Serial.println("[ALERT] -- Humidity above Higher threshold!");

   sendStatusFeedback("Humidity above Higher threshold!", 1);

   if (!manualOverride) {

     thermostat(true, tempLowThreshold + 4);

   }

 }

 if(humidity > humSafe -5 && humidity < humSafe+5){

   Serial.println("[INFO] -- humidity in safe levels!");

   sendStatusFeedback("Humidity in safe levels!", 1);

 }

 delay(500);

 if (ldrOne < ldrOneLowThreshold) {

   Serial.println("[ALERT] -- LDR ONE below Lower threshold!");

   sendStatusFeedback("LDR ONE below Lower threshold!", 1);

   if (!manualOverride) {

     windows(true);
```

```
  }
 }
 if (ldrOne > ldrOneHighThreshold) {
  Serial.println("[ALERT] -- LDR ONE above higher threshold!");
  sendStatusFeedback("LDR ONE above higher threshold!", 1);
  if (!manualOverride) {
   windows(false);
  }
 }
 delay(500);
 if (ldrTwo < ldrTwoLowThreshold) {
  Serial.println("[ALERT] -- LDR TWO below Lower threshold!");
  sendStatusFeedback("LDR TWO below Lower threshold!", 1);
  if (!manualOverride) {
   windows(true);
  }
 }
 if (ldrTwo > ldrTwoHighThreshold) {
  Serial.println("[ALERT] -- LDR TWO above higher threshold!");
  sendStatusFeedback("LDR TWO above higher threshold!", 1);
  if (!manualOverride) {
   windows(false);
  }
 }
 delay(500);
 if (soilMoisture < soilMoistureLowThreshold ) {
  soilSafeState = false;
  Serial.println("[ALERT] -- Soil Moisture  below lower threshold!");
  sendStatusFeedback("Soil Moisture below lower threshold!", 1);
  if (!manualOverride) {
   waterPump(true);
  }
```

```
   }
  if (soilMoisture > soilMoistureHighThreshold ) {
    soilSafeState = false;
    Serial.println("[ALERT] -- Soil Moisture  above higher threshold!");
    sendStatusFeedback("Soil Moisture above higher threshold!", 1);
    if (!manualOverride) {
      waterPump(false);
    }
  }
  delay(500);
  if(soilMoisture > soilSafe -5 && soilMoisture < soilSafe+5){
    soilSafeState = true;
    Serial.println("[INFO] -- Soil Moisture  in safe levels!");
    sendStatusFeedback("soil Moisture in safe levels!", 1);
    delay(500);
    if(soilSafeState && (!manualOverride)) waterPump(false);
  }
  if (pH < pHLowThreshold ) {
    Serial.println("[ALERT] -- pH below Lower threshold!");
    sendStatusFeedback("pH below Lower threshold!", 1);
    sendStatusFeedback(" ", 2);
  }
  if (pH > pHHighThreshold ) {
    Serial.println("[ALERT] -- pH above Higher threshold!");
    sendStatusFeedback("pH above Higher threshold!", 1);
    sendStatusFeedback(" ", 2);
  }
  if(pH > pHSafe -5 && pH < pHSafe+5){
    Serial.println("[INFO] -- PHin safe levels!");
    sendStatusFeedback("pH in safe levels!", 1);
    sendStatusFeedback(" ", 2);
  }
```

```
delay(1000);
}


void setup()
{
  pinMode(WATERPUMP, OUTPUT);
  pinMode(LDR1, INPUT);
  pinMode(WINDOW_MOTOR, OUTPUT);
  // Initialize the Serial Monitor with a baud rate
  Serial.begin(115200);


  // Attach the pin to the servo object
  windowMotor.attach(WINDOW_MOTOR);


  // Initialize the DHT Sensor
  dht.begin();
  // if analog input pin 11 is unconnected, random analog
  // noise will cause the call to randomSeed() to generate
  // different seed numbers each time the sketch runs.
  // randomSeed() will then shuffle the random function.
  // https://www.arduino.cc/reference/en/language/functions/random-numbers/random/
  randomSeed(analogRead(11));
  // The below timers are for reconnecting to the WiFi router and MQTT broker
  mqttReconnectTimer = xTimerCreate("mqttTimer", pdMS_TO_TICKS(2000), pdFALSE, (void*)0, reinter
pret_cast<TimerCallbackFunction_t>(connectToMqtt));
  wifiReconnectTimer = xTimerCreate("wifiTimer", pdMS_TO_TICKS(2000), pdFALSE, (void*)0, reinterp
ret_cast<TimerCallbackFunction_t>(connectToWifi));
  // https://techtutorialsx.com/2019/08/11/esp32-arduino-getting-started-with-wifi-events/
  WiFi.onEvent(WiFiEvent);


  // http://marvinroger.viewdocs.io/async-mqtt-client/2.-API-reference/
  mqttClient.onConnect(onMqttConnect);
```

```
mqttClient.onDisconnect(onMqttDisconnect);
  mqttClient.onSubscribe(onMqttSubscribe);
  mqttClient.onPublish(onMqttPublish);
  mqttClient.setServer(MQTT_HOST, MQTT_PORT);
  mqttClient.onMessage(onMqttMessage);
  // If your broker requires authentication.
  mqttClient.setCredentials(MQTT_USERNAME, MQTT_PASS);
  connectToWifi();
  sendStatusFeedback("ALL OK!", 2);
}

void loop() {
  unsigned long currentMillis = millis();
  // Every X number of seconds (interval = 10 seconds)
  // it publishes a new MQTT message
  if (currentMillis - previousMillis >= 2000) {
    //Save the last time a new message was sent.
    previousMillis = currentMillis;
    getSensorData(false);
    Serial.printf("[ACTION] -- Publishing temperature\n");
    Serial.printf("[INFO] -- Temperature:%d\n", temperature);
    mqttClient.publish(MQTT_PUB_TEMP, 1, true, String(temperature).c_str());
    delay(500);

    Serial.printf("[ACTION] -- Publishing humidity\n");
    Serial.printf("[INFO] -- Humidity:%d\n", humidity);
    mqttClient.publish(MQTT_PUB_HUM, 1, true, String(humidity).c_str());
    delay(500);
    Serial.printf("[ACTION] -- Publishing soil moisture\n");
    Serial.printf("[INFO] -- Soil Moisture:%d\n", soilMoisture);
    mqttClient.publish(MQTT_PUB_SOIL_MOISTURE, 1, true, String(soilMoisture).c_str());
    delay(500);
```

```cpp
    Serial.printf("[ACTION] -- Publishing ldr one\n");
    Serial.printf("[INFO] -- LDR ONE:%d\n", ldrOne);
    mqttClient.publish(MQTT_PUB_LDR1, 1, true, String(ldrOne).c_str());
    delay(500);


    Serial.printf("[ACTION] -- Publishing ldr two\n");
    Serial.printf("[INFO] -- LDR TWO:%d\n", ldrTwo);
    mqttClient.publish(MQTT_PUB_LDR2, 1, true, String(ldrTwo).c_str());
    delay(500);


    Serial.printf("[ACTION] -- Publishing pH\n");
    Serial.printf("[INFO] -- pH:%d\n", pH);
    mqttClient.publish(MQTT_PUB_PH_LEVEL, 1, true, String(pH).c_str());
    delay(500);
  }
  delay(3000);
  setSafeValues();
  thresholdChecks();
}


void waterPump(int state) {
  if (state) {
    Serial.println("[ACTION] -- MOTOR ON");
    digitalWrite(WATERPUMP, HIGH);
    sendStatusFeedback("MOTOR ON", 2);
    delay(2000);
  }
  else {
    Serial.println("[ACTION] -- MOTOR OFF");
    digitalWrite(WATERPUMP, LOW);
    sendStatusFeedback("MOTOR OFF", 2);
```

```cpp
delay(2000);
 }
}

void windows(int state) {
 if (state) {
  Serial.println("[ACTION] -- OPENING WINDOWS");
  delay(2000);
  windowMotor.write(180);
  sendStatusFeedback("OPENING WINDOWS", 2);
  delay(2000);
 }
 else {
  Serial.println("[ACTION] -- CLOSING WINDOWS");
  delay(2000);
  windowMotor.write(0);
  sendStatusFeedback("CLOSING WINDOWS", 2);
  delay(2000);
 }
}

void thermostat(int state, int temp) {
 if (state) {
  Serial.println("[ACTION] -- TURNING ON THERMOSTAT");
  Serial.printf("Setting temperature to:%d", temp);
  sendStatusFeedback("Setting thermostat to " + String(temp) + "°C", 2);
 }
 else {
  Serial.println("[ACTION] -- TURNING OF THERMOSTAT");
  sendStatusFeedback("Thermostat off", 2);
 }
}
```

3.6 – APPLICATIONS

People are still working on different Smart Farming technology using IoT, so the anticipated benefits of this technology is, Remote monitoring for farmers, water and other natural resource conservation, good management also allows improved livestock farming, the things which are not visible to naked eye can be seen resulting is accurate farmland and crop evaluation, good quality as well as improved quantity, the facility to get the real- time data for useful insights.

More over this prototype can be deployed in small green houses and especially can be used for growing rare crops which requires critical monitoring.

People who are interested in growing crops but their hectic schedules may not allow them to grow crops because they require constant monitoring. So, this autonomous crop monitoring system can be used to know everything that is happening to the crop and around they are monitored using sensors and the data is sent to the user's mobile. This facilitates the user to know what is happening and can control the actuators that can sprinkle water or open the windows or something the crop needs.

3.7 – ADVANTAGES

- **Efficiency:**
  - Since the overall environment is a controlled environment, crops can be grown in an efficient manner and any abnormality can be negated.

- **Yield:**
  - A positive growth rate can be observed because the crop monitoring system provides everything a crop wants and every critical condition around the crop is monitored.

- **Condition Monitoring:**
  - The factors effecting the crop are constantly monitored. For example, factors such as temperature, humidity, soil moisture, pH, luminosity which are critical for plant's growth are kept track.

- **Application:**
  - The App helps in providing the sensor data and a user action panel where the user can decide what the MCU must perform manually.
  - The App can also toggle automatic mode where ESP32 will take care of everything and will perform best action that is required at that particular point of time.

3.8 – DISADVANTAGES

- Sensors of higher accuracy can be used but the cost increases drastically.
- The smart farming based technology requires farmer to learn about it. This is the major challenge in adopting smart agricultural farming at large scale across the countries.
- Faulty sensor reading can cause releasing of more water or no water at all. The system purely depends on the quality of the sensor monitoring. So the system is not fault tolerant.

CHAPTER 4 – OPERATING PROCEDURE

- Upload the program to the ESP32 using Arduino IDE. Check Figure iv - Chapter 4 – Upload.

- Start the mosquitto service on the system where you hosted the mosquitto broker using the command "mosquitto –v". Check Figure iv - Chapter 4 - Mosquitto Command.

- The above command will start the mosquitto service in verbose mode. This will help us in tracking what operation is being done and what packets are being transferred.

- Then open the app "MQTT Dashboard" set the credentials and server ip in the settings options of the app. Check Figure vi - Chapter 4 - App Settings.

- The app will then connect to the server that is hosted locally on a static ip.

- In the app we can set the threshold settings for each parameter that we want to monitor.

- User Actions can be triggered by activating the manual override mode.

- Actions such as turning on the water pump, closing and opening windows can be done.

- Additional actions can be programmed, such as controlling the thermostat and different modes of the water pump.
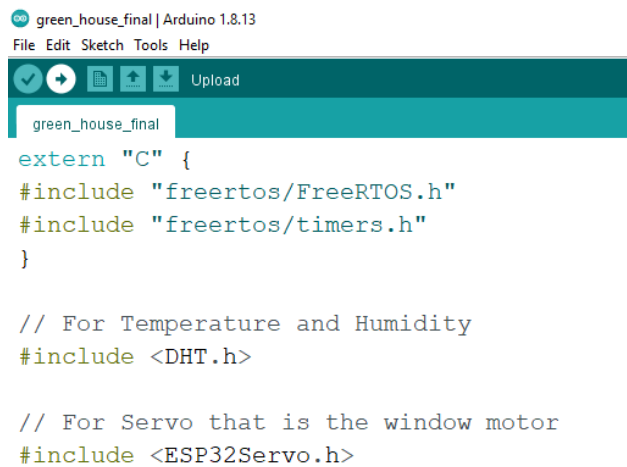
Figure xii - Chapter 4 – Upload
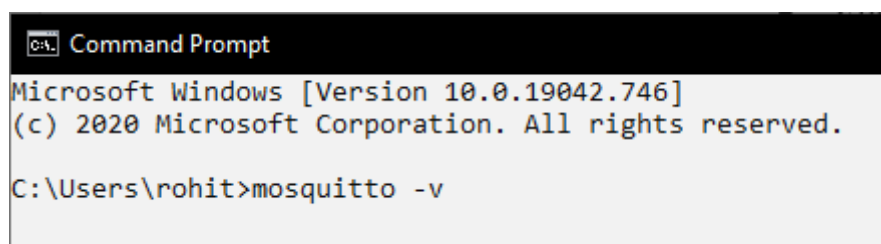
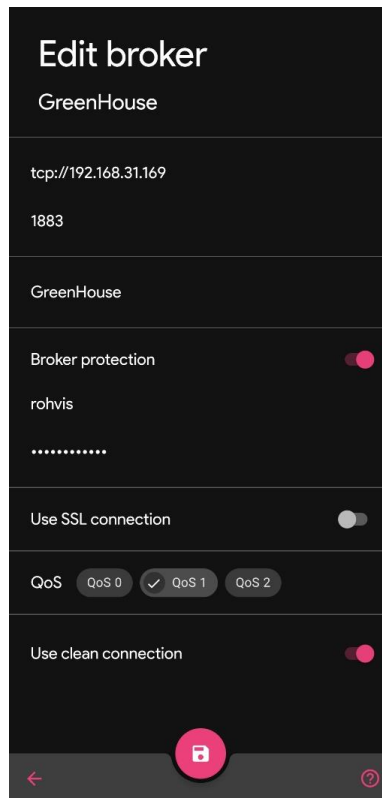Figure xiii - Chapter 4 - Mosquitto Command

Figure xiv - Chapter 4 - App Settings

- **tcp://192.168.31.169** is the static ip where the server is hosted
- **1883** is the port that it is listening to.
- In the **Broker protection,** we provide our MQTT credentials in-order to authenticate to the mosquitto broker.
- **QoS** is set to level 1 The **Quality of Service** (QoS) level is an agreement between the sender of a message and the receiver of a message that defines the guarantee of delivery for a specific message.
- **QoS level 1** guarantees that a message is delivered at least one time to the receiver. The sender stores the message until it gets a PUBACK packet from the receiver that acknowledges receipt of the message. It is possible for a message to be sent or delivered multiple times.
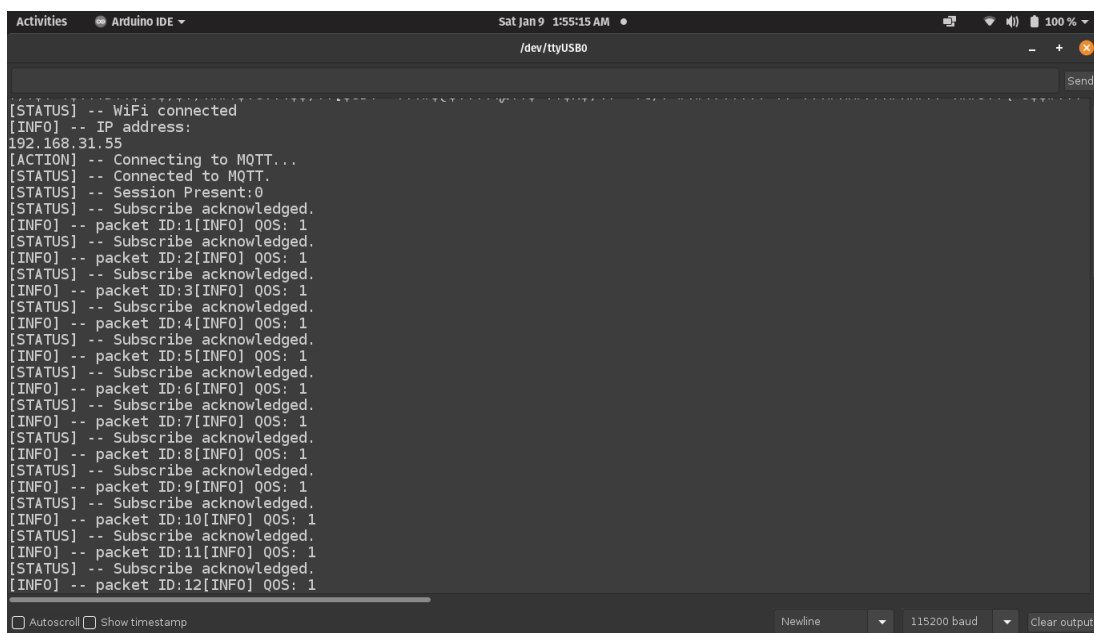
Figure xv - Chapter 4 - QoS 1

CHAPTER 5 – RESULTS

- Efficiency in the growth of the plant and higher yield is observed when compared to previous growth rate and yield rate.

- The App functionalities are observed to be accurate and the action performed is user intended.

- Power consumption is low and by using the battery of capacity 10000mAh has shown the overall working period of 10 days.

- Every action performed by the ESP32 is logged perfectly and the log is reflected in the App too.

- Response period of every action initiated from the app took a time of 1 second to register in mosquitto broker.

- Actuation of action by the ESP32 took a period of 2 seconds (signal send from mosquitto broker + signal send from ESP32 to actuator + actuator delays in the code).

5.1 – SCREENSHOTS

Figure xvi - Chapter 5.1 - Serial Monitor of ESP32

o The first segment of the App is the Sensors Readings. Here the data sensed from the sensors is sent via the MQTT broker.

o In the second segment of the App is the threshold settings. Here the user can set individual threshold limits for each parameter sensed by the sensor.

o In the third segment of the App is the User Actions. Here the user can enable **Manual Override.**

o This means the user can perform against the action suggested by the ESP32 or proceed with its decision.

o If the **Manual Override** options is turned off, ESP32 will perform the best action possible to neutralize the situation.

o If we observe there are two status boards in the User Actions section.

o **Feedback Status** displays the status of the control operations that are being done by ESP32.

o **Last Action Status** displays the action performed by ESP32 and both the statuses are logged in the **Events Log**.
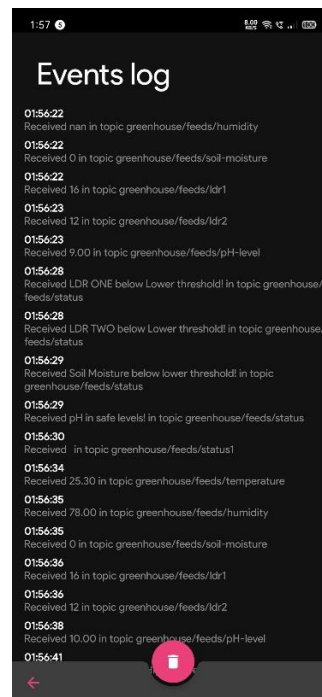

Figure xvii - Chapter 5.1 - Events Log

Figure xviii - Chapter 5.1 - App UI

CHAPTER 6 – CONCLUSION

Thus, the project proposes an idea of combining the latest technology into the agricultural field to turn the traditional methods of irrigation to modern methods thus making easy productive, and economical cropping. Some extent of automation is introduced enabling the concept of monitoring the field and the crop conditions within some long-distance ranges using mosquitto brokers. The advantages like water saving and labour-saving are initiated using sensors that work automatically as they are programmed. This concept of modernization of agriculture is simple, affordable and operable.

CHAPTER 7 – FUTURE SCOPE

- Later, it can be interfered with HYDROPHONICS which is hydro-irrigation method (requires no soil) for complete transformation of phase of Irrigation.

- Every other person can monitor condition of the filed by working at their own places without being present in the field, thus encouraging agriculture.

- The camera module can be placed on a drone to capture huge number of fields at once by flying in the air both horizontally and vertically such that every look and corner of the plant is visible and intruders, plant diseases can be detected using machine learning and AI.

REFERENCES

[1] Balaji Bhanu, Raghava Rao, J.V.N. Ramesh and Mohammed Ali hussain,"Agriculture Field Monitoring and Analysis using Wireless SensorNetworks for improving Crop Production", 2014 Eleventh InternationalConference on Wireless and Optical Communications Networks(WOCN)

[2] LIU Dan, Cao Xin, Huang Chongwei, JI Liang Liang, "Intelligent agentgreenhouse environment monitoring system based on IOT technology",2015 International Conference on Intelligent Transportation, Big Data &Smart City

[3] Joseph Haule, Kisangiri Michael, "Deployment of wireless sensornetworks (WSN) in automated irrigation management and schedulingsystems: a review", Science, Computing and Telecommunications(PACT), 2014, Pan African Conference