PROJECT REPORT

Smart Flower Pot.

Name :- P.I.D.De Silva

Index No :- AS2021914

Abstract

This project explores the development of a Smart Flower Pot designed to enhance plant care through automation and IoT integration. The system is built using a microcontroller and multiple sensors to monitor soil moisture, enabling automated watering for optimal plant health. Additionally, a WiFi module connects the system to the Blynk IoT platform, allowing users to monitor real-time data and control the flower pot remotely via a mobile application. By combining automation with remote accessibility, this smart flower pot aims to reduce the maintenance burden on users, ensuring plants receive consistent care regardless of the user's schedule or availability. The project demonstrates the potential of smart systems in everyday applications and provides a foundation for further development in automated plant care.

Contents

1. Introduction.

Maintaining healthy plants can be challenging, especially for those with busy schedules or limited knowledge of proper plant care. Factors such as watering frequency, soil moisture, and environmental conditions all play critical roles in a plant's well-being. However, these requirements are easy to overlook in daily life, leading to under- or over-watering, which can damage plant health. This issue has prompted the need for automated solutions that can simplify plant maintenance while providing the care plants need to thrive.

The Smart Flower Pot project addresses this need by integrating advanced sensor technology and IoT capabilities into a single, user-friendly system. Equipped with a soil moisture sensor, automated watering mechanism, and a WiFi module, the smart flower pot can autonomously monitor moisture levels in the soil and water the plant when necessary. Through the Blynk IoT platform, users can also view real-time data and control the system remotely via a mobile app, ensuring plants are consistently cared for even when users are away.

A unique feature of this smart flower pot is its OLED screen, which provides a human-like "face" that interacts directly with users. By displaying friendly expressions or plant health information, the screen not only makes the system more engaging but also provides a sense of comfort and mental relaxation. This direct connection with users offers an extra benefit, making plant care not just a routine task but an enjoyable interaction.

2. Aim of the project.

The primary aim of this project is to create a Smart Flower Pot that combines plant care automation with user engagement features to improve the overall experience of plant maintenance. This objective is broken down into two main areas.

2.1 Improving plant care.

The Smart Flower Pot is designed to monitor and maintain optimal soil moisture levels for healthy plant growth. By using sensors and an automated watering system, it ensures that plants receive the right amount of water at the right time, reducing the risk of under- or over-watering. The integration with the Blynk IoT platform allows users to monitor and control their plants' environment remotely, making plant care more accessible and reliable, even when users are away.

2.2 Mental Relaxation.

Beyond practical plant maintenance, the project aims to offer a relaxing and enjoyable experience for users through the use of an OLED screen. This display features a friendly face that engages with users, providing a comforting presence and enhancing user interaction. The "face" creates an emotional connection, making plant care more than a chore, as it offers mental relaxation and a sense of companionship, which can be especially beneficial in indoor environments.

3.System Requirements Specification.

3.1 Functional Requirements.

- Soil Moisture Monitoring: The system should continuously measure soil moisture levels using a sensor and store data for analysis.

- Automated Watering: The "Automatic Watering" function is designed to maintain optimal soil moisture levels in the smart flower pot. When the soil moisture level below 30%, the ESP32 board automatically activates the motor, which turns on the water pump to hydrate the plant. As soon as the soil moisture rises above 30%, the motor and pump automatically shut off, stopping the watering. Throughout this process, the user can monitor the soil moisture levels in real-time via the Blynk IoT app. When conditions are normal, the OLED display returns to its usual appearance.

- Real-Time Data Monitoring: The system must connect to the Blynk IoT platform to allow users to monitor moisture levels and control watering remotely.

- OLED Display for User Interaction: The system should display a friendly "face" on an OLED screen that responds to plant conditions, providing mental relaxation for users.

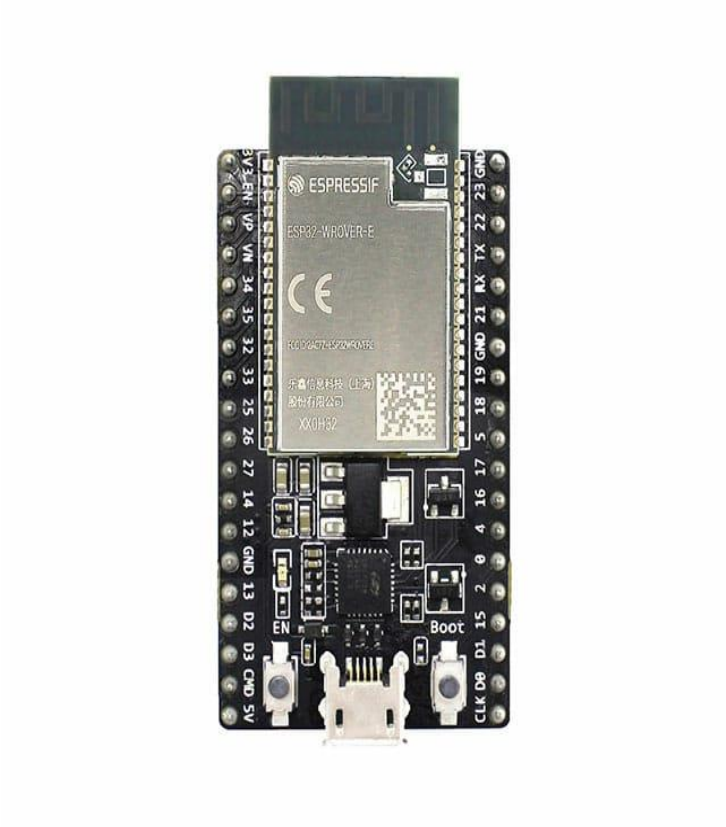
3.2 Non-Functional Requirements.

- Water Level Monitoring: The "Water Level Monitoring" function ensures that the plant never runs out of water. When the water level in the reservoir drops below 10%, the OLED screen displays "I'm thirsty!" as a reminder for the user to manually refill the water. Once the water is replenished, the display returns to its normal appearance. Additionally, the user can monitor the water level at any time through the Blynk IoT app, making it easy to keep the reservoir topped up and the plant well-hydrated.

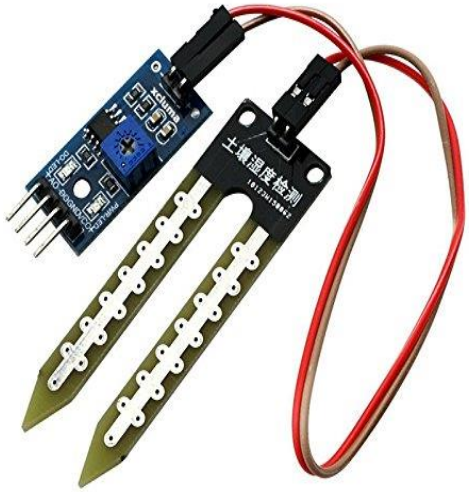
3.3 Hardware Requirements.

- ❖ Microcontroller.
- ❖ Soil Moisture Sensor.
- ❖ DHT11 Sensor.
- ❖ Water Level Sensor.
- ❖ Relay Module.
- ❖ Water Pump.
- ❖ OLED Screen.
- ❖ Digital Display.
- ❖ Power Supply.

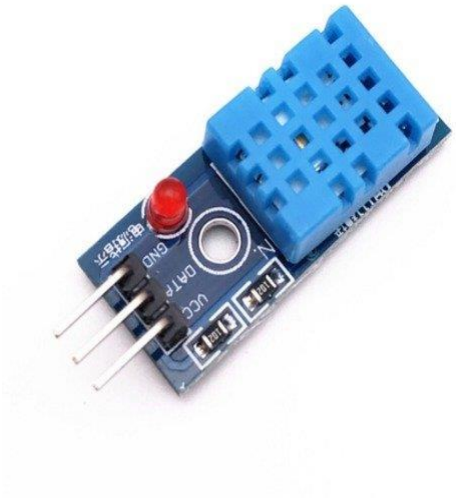
3.3.1 Microcontroller: I used to ESP32 board as my microcontroller to handle data processing and control tasks.



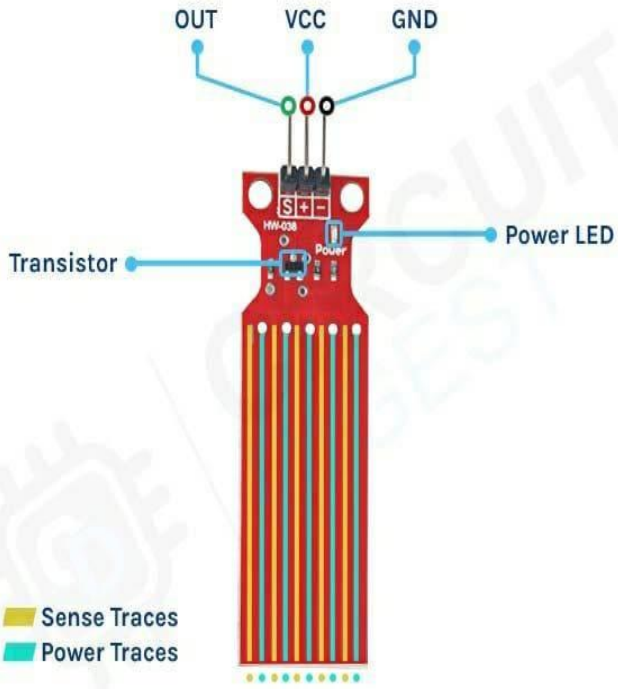
3.3.2 Soil Moisture Sensor: A sensor to measure the moisture level in the soil and send the data to the microcontroller.



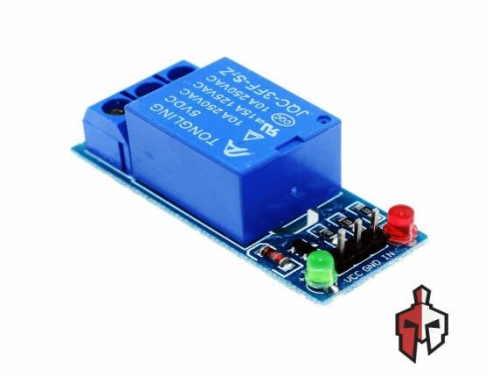
3.3.3 DHT11 Sensor: A Humidity and Temperature sensor measure the temperature in the soil and send the data to the microcontroller.



3.3.4 Water Level Sensor: A sensor to measure the water level in the soil as a percentage and send the data to the microcontroller.



3.3.5 Relay Module: Displays essential data like soil moisture, temperature, and water level, providing immediate feedback on the plant's status.



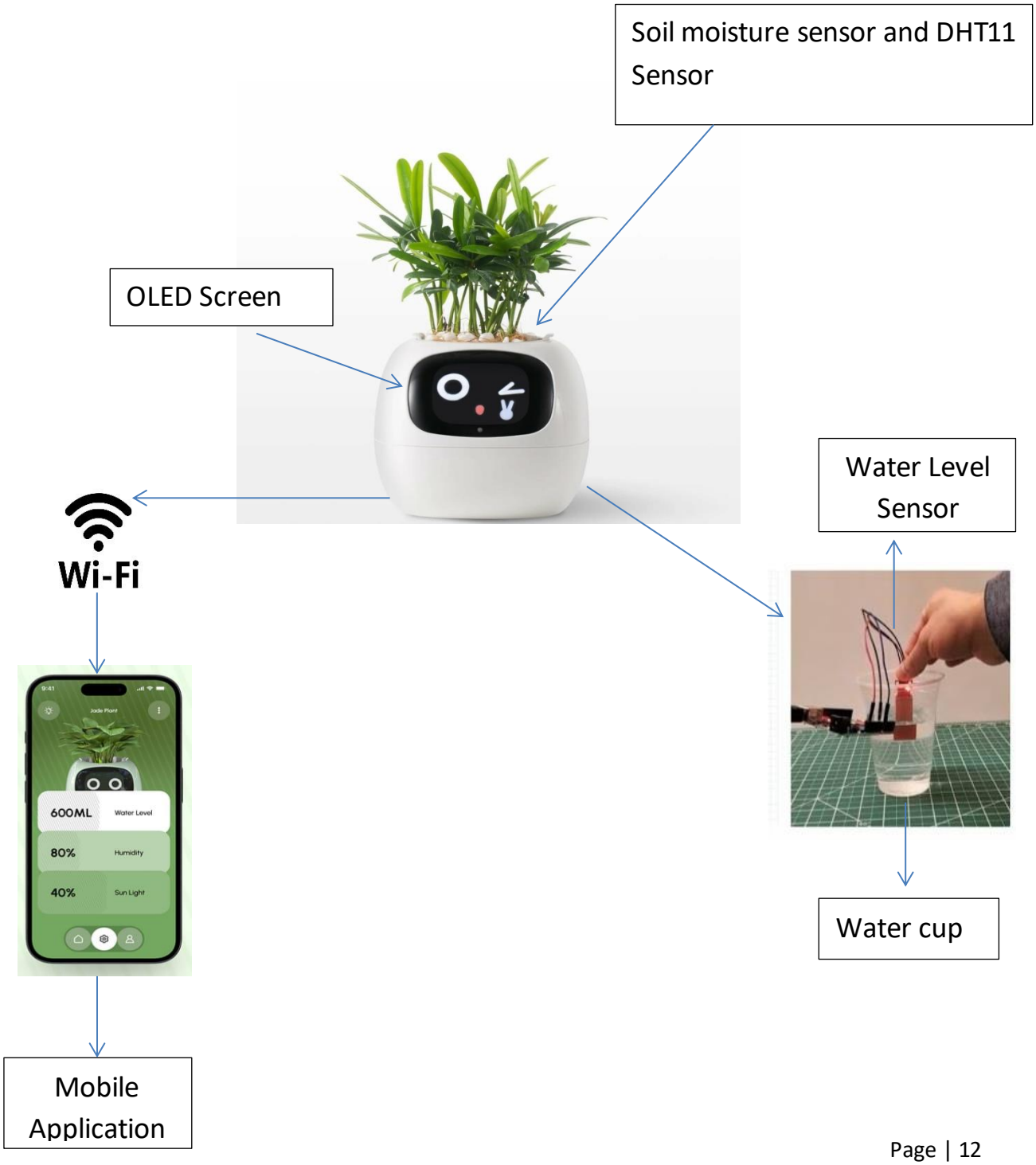
3.3.6 Water Pump: A small water pump to irrigate the plant as needed based on moisture readings.

3.3.7 OLED Screen: An OLED display to show a user-friendly "face" that interacts with users and enhances engagement.

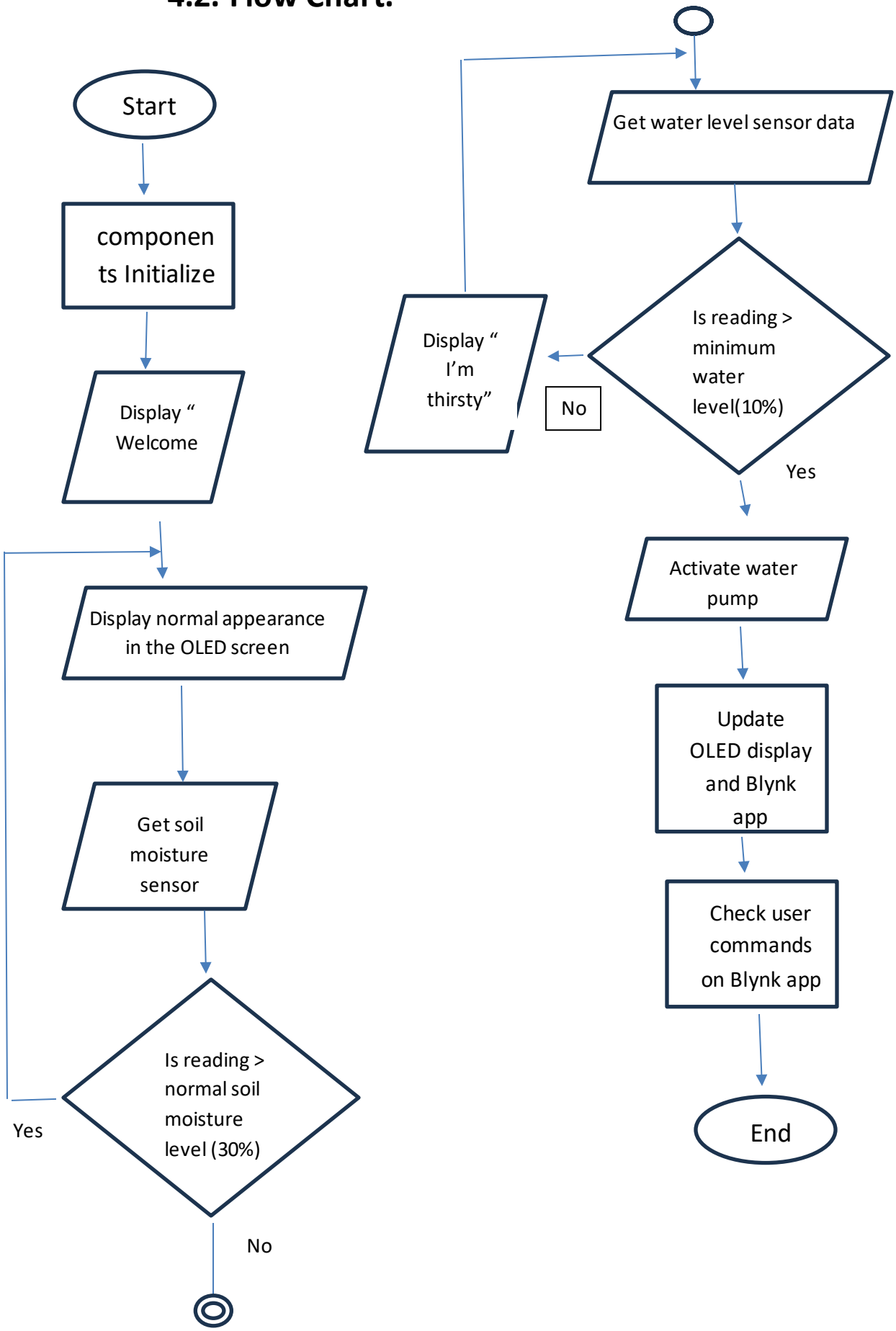
3.3.9 Power Supply: As a prototype my laptop operates microcontroller , sensors and displays . Water pump has a battery pack to operate it.

4. System Analysis and Design.

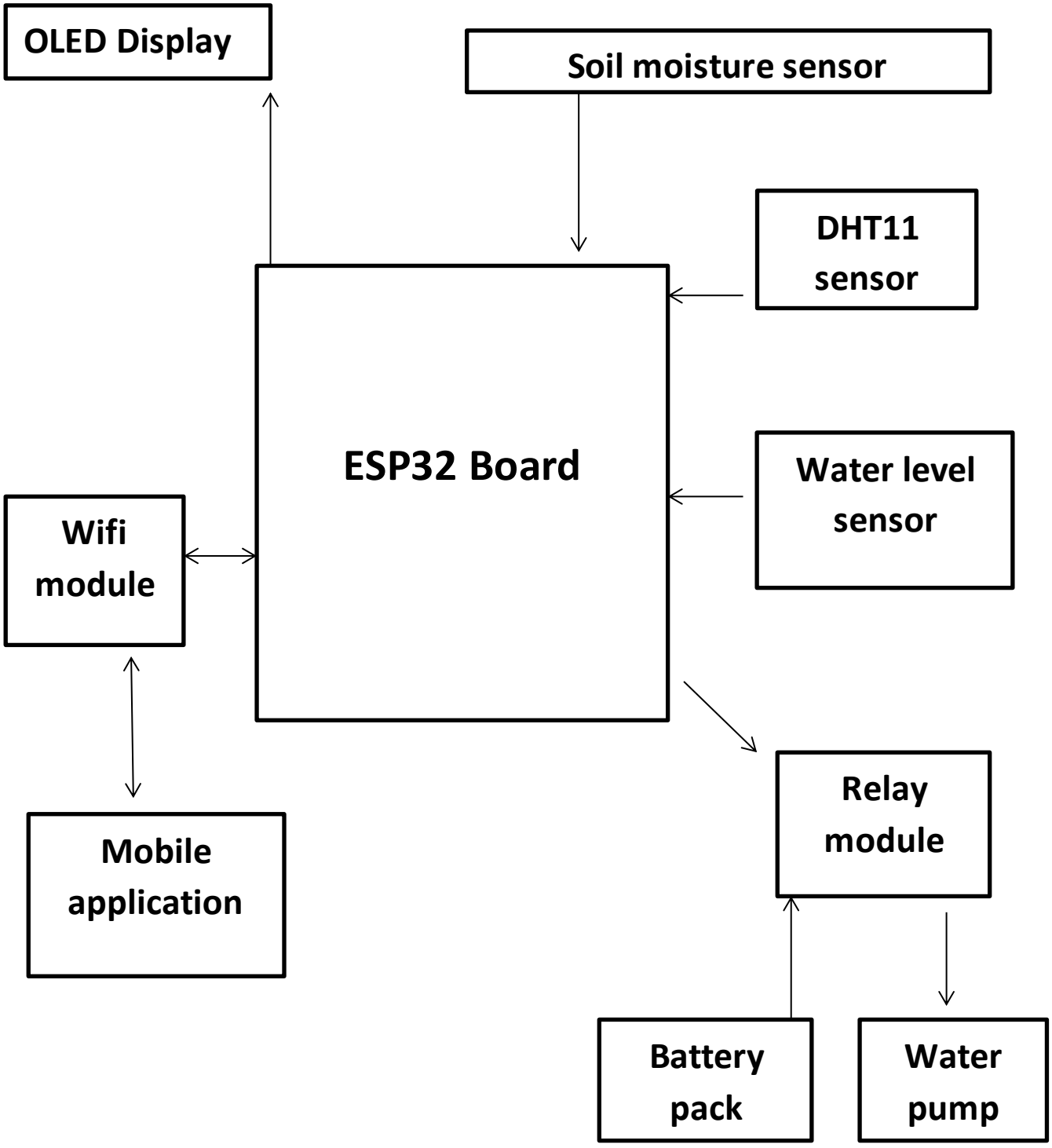
4.1 System Design.



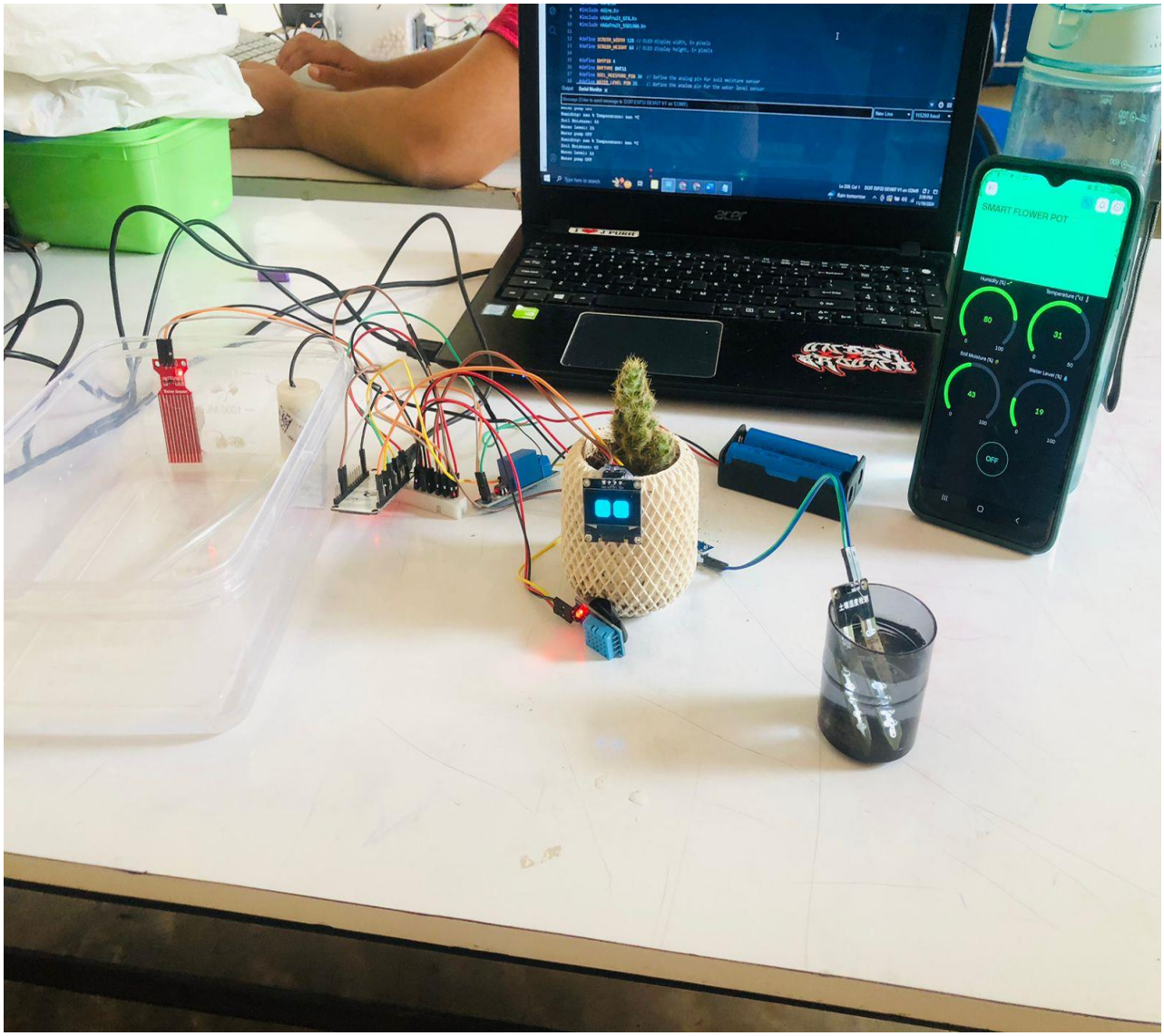
4.2. Flow Chart.



4.3 Block Diagram.



4.5 Physical Review.



4.5 Budget.

ITEM	PRICE (RS.)
SOIL MOISTURE SENSOR	**200**
DHT11 SENSOR	**230**
WATER LEVEL SENSOR	**150**
OLED DISPLAY	**630**
LCD DISPLAY	**680**
RELAY MODULE	**350**
WATER PUMP	**190**
BATTERY PACK	**200**
JUMPER WIRES	**450**
TOTLE	**3,080**

5. Implementation.

5.1 Source Code.

```
#define BLYNK_TEMPLATE_ID "TMPL6fM0eg9Mn"

#define BLYNK_TEMPLATE_NAME "smart flower pot"

#define BLYNK_AUTH_TOKEN "NkhWzzLl1YNStpnsPlmx5QV7pSFc-YRb"

#include <WiFi.h>

#include <BlynkSimpleEsp32.h>

#include "DHT.h"

#include <SPI.h>

#include <Wire.h>

#include <Adafruit_GFX.h>

#include <Adafruit_SSD1306.h>
```

```
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

#define DHTPIN 4
#define DHTTYPE DHT11
#define SOIL_MOISTURE_PIN 34  // Define the analog pin for soil moisture sensor
#define WATER_LEVEL_PIN 35    // Define the analog pin for the water level sensor
#define RELAY_PIN 5           // Define the GPIO pin for the relay module


#define OLED_RESET    -1 // Reset pin # (or -1 if sharing Arduino reset pin)
#define SCREEN_ADDRESS  0x3C // Address for 128x64 OLED display
Adafruit_SSD1306 display(SCREEN_WIDTH,  SCREEN_HEIGHT,  &Wire, OLED_RESET);
DHT dht(DHTPIN, DHTTYPE);

const char* ssid = "Ishini";
const char* password = "12344321";

// Reference state for eyes
int ref_eye_height = 40;
int ref_eye_width = 40;
int ref_space_between_eye = 10;
int ref_corner_radius = 10;

// Current state of the eyes
int left_eye_height = ref_eye_height;
int left_eye_width = ref_eye_width;
int left_eye_x = 32;
```

```
int left_eye_y = 32;
int right_eye_x = 32 + ref_eye_width + ref_space_between_eye;
int right_eye_y = 32;
int right_eye_height = ref_eye_height;
int right_eye_width = ref_eye_width;

void draw_eyes(bool update = true) {
  display.clearDisplay();
  // Draw left eye
  int x = int(left_eye_x - left_eye_width / 2);
  int y = int(left_eye_y - left_eye_height / 2);
  display.fillRoundRect(x,  y, left_eye_width, left_eye_height, ref_corner_radius, SSD1306_WHITE);
  // Draw right eye
  x = int(right_eye_x - right_eye_width / 2);
  y = int(right_eye_y - right_eye_height / 2);
  display.fillRoundRect(x,  y, right_eye_width, right_eye_height, ref_corner_radius, SSD1306_WHITE);
  if (update) {
    display.display();
  }
}

void center_eyes(bool update = true) {
  // Move eyes to the center of the display
  left_eye_x = SCREEN_WIDTH / 2 - ref_eye_width / 2 - ref_space_between_eye / 2;
  left_eye_y = SCREEN_HEIGHT / 2;
  right_eye_x = SCREEN_WIDTH / 2 + ref_eye_width / 2 + ref_space_between_eye / 2;
  right_eye_y = SCREEN_HEIGHT / 2;
  draw_eyes(update);
}
```

```
void blink(int speed = 12) {
  draw_eyes();
  for (int i = 0; i < 3; i++) {
    left_eye_height -= speed;
    right_eye_height -= speed;
    draw_eyes();
    delay(1);
  }
  for (int i = 0; i < 3; i++) {
    left_eye_height += speed;
    right_eye_height += speed;
    draw_eyes();
    delay(1);
  }
}

void sleep() {
  left_eye_height = 2;
  right_eye_height = 2;
  draw_eyes(true);
}

void wakeup() {
  sleep();
  for (int h = 0; h <= ref_eye_height; h += 2) {
    left_eye_height = h;
```

```
      right_eye_height = h;
      draw_eyes(true);
    }
  }

void happy_eye() {
  center_eyes(false);
  // Draw inverted triangle over lower part of eye
  int offset = ref_eye_height / 2;
  for (int i = 0; i < 10; i++) {
    display.fillTriangle(left_eye_x  - left_eye_width / 2 - 1, left_eye_y + offset,
                left_eye_x + left_eye_width / 2 + 1, left_eye_y + 5 + offset,
                left_eye_x - left_eye_width / 2 - 1, left_eye_y + left_eye_height + offset, SSD1306_BLACK);
    display.fillTriangle(right_eye_x  + right_eye_width / 2 + 1, right_eye_y + offset,
                right_eye_x - left_eye_width / 2 - 1, right_eye_y + 5 + offset,
                right_eye_x + right_eye_width / 2 + 1, right_eye_y + right_eye_height + offset,
SSD1306_BLACK);
    offset -= 2;
    display.display();
    delay(1);
  }
  display.display();
  delay(1000);
}

void saccade(int  direction_x, int direction_y) {
  int direction_x_movement_amplitude = 8;
  int direction_y_movement_amplitude = 6;
  int blink_amplitude  = 8;
```

```
  for (int i = 0; i < 1; i++) {
    left_eye_x += direction_x_movement_amplitude * direction_x;
    right_eye_x += direction_x_movement_amplitude * direction_x;
    left_eye_y += direction_y_movement_amplitude * direction_y;
    right_eye_y += direction_y_movement_amplitude * direction_y;
    right_eye_height -= blink_amplitude;
    left_eye_height -= blink_amplitude;
    draw_eyes();
    delay(1);
  }
  for (int i = 0; i < 1; i++) {
    left_eye_x += direction_x_movement_amplitude * direction_x;
    right_eye_x += direction_x_movement_amplitude * direction_x;
    left_eye_y += direction_y_movement_amplitude * direction_y;
    right_eye_y += direction_y_movement_amplitude * direction_y;
    right_eye_height += blink_amplitude;
    left_eye_height += blink_amplitude;
    draw_eyes();
    delay(1);
  }
}

void move_big_eye(int direction) {
  int direction_oversize = 1;
  int direction_movement_amplitude = 2;
  int blink_amplitude = 5;

  for (int i = 0; i < 3; i++) {
```

```
    left_eye_x += direction_movement_amplitude * direction;
    right_eye_x += direction_movement_amplitude * direction;
    right_eye_height -= blink_amplitude;
    left_eye_height -= blink_amplitude;
    if (direction > 0) {
      right_eye_height += direction_oversize;
      right_eye_width += direction_oversize;
    } else {
      left_eye_height += direction_oversize;
      left_eye_width += direction_oversize;
    }
    draw_eyes();
    delay(1);
  }
  for (int i = 0; i < 3; i++) {
    left_eye_x += direction_movement_amplitude * direction;
    right_eye_x += direction_movement_amplitude * direction;
    right_eye_height += blink_amplitude;
    left_eye_height += blink_amplitude;
    if (direction > 0) {
      right_eye_height += direction_oversize;
      right_eye_width += direction_oversize;
    } else {
      left_eye_height += direction_oversize;
      left_eye_width += direction_oversize;
    }
    draw_eyes();
    delay(1);
  }
```

```
delay(1000);


for (int i = 0; i < 3; i++) {
  left_eye_x -= direction_movement_amplitude * direction;
  right_eye_x -= direction_movement_amplitude * direction;
  right_eye_height -= blink_amplitude;
  left_eye_height -= blink_amplitude;
  if (direction > 0) {
    right_eye_height -= direction_oversize;
    right_eye_width -= direction_oversize;
  } else {
    left_eye_height -= direction_oversize;
    left_eye_width -= direction_oversize;
  }
  draw_eyes();
  delay(1);
}
for (int i = 0; i < 3; i++) {
  left_eye_x -= direction_movement_amplitude * direction;
  right_eye_x -= direction_movement_amplitude * direction;
  right_eye_height += blink_amplitude;
  left_eye_height += blink_amplitude;
  if (direction > 0) {
    right_eye_height -= direction_oversize;
    right_eye_width -= direction_oversize;
  } else {
    left_eye_height -= direction_oversize;
    left_eye_width -= direction_oversize;
```

```
      }
      draw_eyes();
      delay(1);
    }

    center_eyes();
}

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

  pinMode(RELAY_PIN, OUTPUT);
  digitalWrite(RELAY_PIN, HIGH);  // Initialize relay as off (assuming active LOW relay)

  WiFi.begin(ssid, password);

  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("Connected to Wi-Fi.");
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
  } else {
    Serial.println("Failed to connect to Wi-Fi.");
  }

  Serial.println("Connecting to Wi-Fi and Blynk...");
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, password);
    display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);
```

```
  display.display();
  delay(2000); // Pause for 2 seconds

  // Clear the buffer
  display.clearDisplay();
}

void loop() {

  // Animation sequence
  Blynk.run();  // Handle Blynk connection and communication

 delay(2000);

 float humidity = dht.readHumidity();
 float temperature = dht.readTemperature();
 int soilMoistureValue = analogRead(SOIL_MOISTURE_PIN);
 int waterLevelValue = analogRead(WATER_LEVEL_PIN);
 int motor;
 // Convert sensor readings to percentages
 soilMoistureValue = map(soilMoistureValue,  0, 4095, 100, 0);
 waterLevelValue = map(waterLevelValue, 0, 4095, 0,100);

 // Print readings to Serial Monitor
 Serial.print("Humidity: ");
 Serial.print(humidity);
 Serial.print(" %\t");
 Serial.print("Temperature: ");
 Serial.print(temperature);
```

```
Serial.println(" *C");


Serial.print("Soil Moisture: ");
Serial.println(soilMoistureValue);


Serial.print("Water Level: ");
Serial.println(waterLevelValue);
 if (soilMoistureValue < 30) {  // Threshold value for low soil moisture
  digitalWrite(RELAY_PIN, LOW);  // Turn on the water pump (assuming active LOW relay)
  Serial.println("Water pump ON");
  int motor =1;


} else {
  digitalWrite(RELAY_PIN, HIGH);  // Turn off the water pump
  Serial.println("Water pump OFF");
 int motor = 0;
}


// Send readings to Blynk app
Blynk.virtualWrite(V1, temperature);       // Send temperature to Virtual Pin V1
Blynk.virtualWrite(V2, humidity);          // Send humidity to Virtual Pin V2
Blynk.virtualWrite(V3, soilMoistureValue);  // Send soil moisture to Virtual Pin V3
Blynk.virtualWrite(V4, waterLevelValue);    // Send water level to Virtual Pin V4
Blynk.virtualWrite(V0, motor);
if (waterLevelValue < 10) {


  wakeup();
  delay(1000);
  happy_eye();
```

```
  delay(1000);
  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(40, 0);
  display.println("I'm  ");
  display.setCursor(20, 20);
  display.println("thersty!");

  display.display();

}
else {

  wakeup();
  delay(1000);

  blink();
  delay(1000);

  happy_eye();
  delay(1000);

  saccade(1,  0); // Move right
  delay(1000);

  saccade(-1,  0); // Move left
  delay(1000);
```

```
    move_big_eye(1); // Move big right eye
    delay(1000);

    move_big_eye(-1); // Move big left eye
    delay(1000);

    sleep();
    delay(1000);
  }
}
```

5.2 Results.

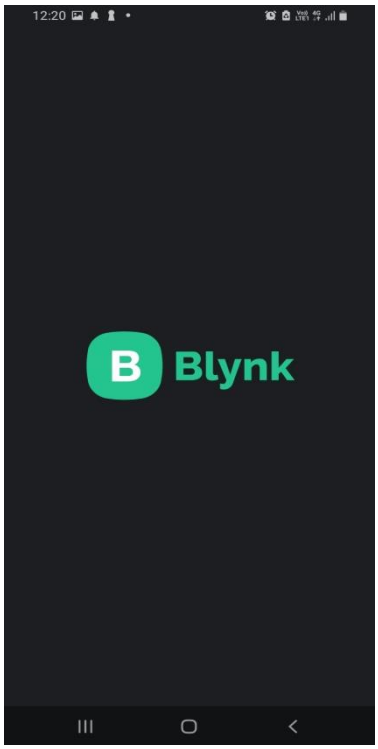
❖ *Successful Soil Moisture Monitoring.*

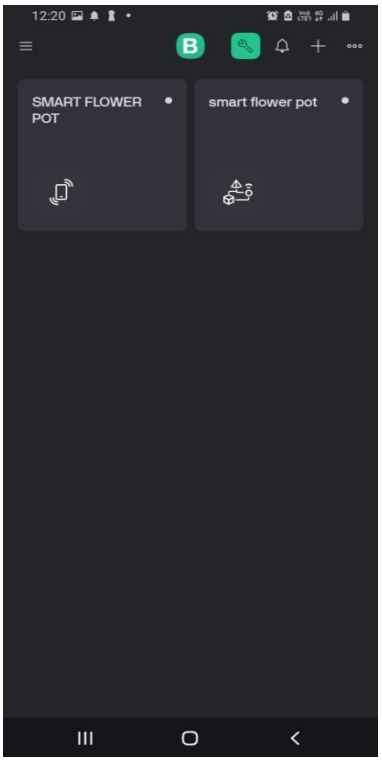
The soil moisture sensor identify the minimum moisture level (30%) in the soil and transfer data to the controller.

❖ *Automated Watering Activation.*

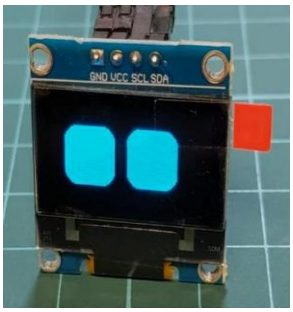
The system accurately triggers the water pump based on predefined moisture threshold (30%).

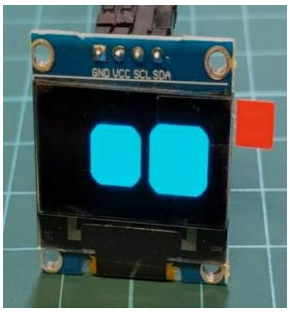
❖ *Real-Time Data Monitoring on Blynk IOT App.*

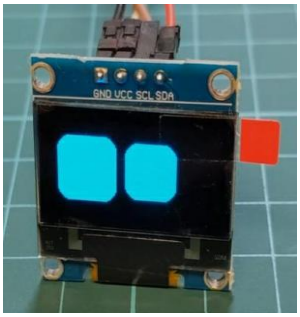


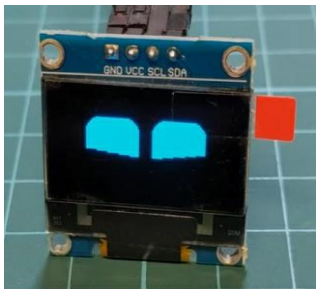


❖ *OLED Display Interaction.*









❖ *User Feedback and Mental Relaxation Benefits.*

The OLED screen received positive feedback for its engaging and user-friendly design. Users found the "face" animations comforting, adding a sense of companionship and reducing stress. The interactive features motivated users to care for their plants more actively. Suggestions included adding customizable expressions for a more personalized experience. Overall, the OLED screen enhanced user satisfaction and provided mental relaxation alongside plant care.

6. Conclusion and Future Scope.

6.1 Conclusion.

The Smart Flower Pot project successfully demonstrates how IoT and automation can simplify plant care, making it accessible and efficient for users with varying levels of experience. By combining a soil moisture sensor, automated watering system, and remote monitoring capabilities through the Blynk platform, the project addresses common challenges in plant maintenance , ensuring plants receive the necessary care even when users are not present. The inclusion of an OLED screen that displays a friendly face adds an

engaging, user-centered element, providing mental relaxation and a unique interactive experience.

6.2 *Future Scope.*

1. Enhanced Environmental Monitoring

Future versions of the Smart Flower Pot could include additional sensors for light and nutrients . This would allow the system to provide more comprehensive data on environmental conditions, helping users better understand the ideal care settings for their plants.

2.Integration with AI for Plant Care Recommendations

By incorporating AI, the system could analyze historical data and environmental trends to give users personalized care recommendations, such as ideal watering frequency or light levels for specific plant types. AI could also predict when the plant might need watering, based on weather patterns and indoor conditions.

3.Automatic Fertilization System

Adding a feature for automated fertilization would further enhance plant health. A dual-tank system could handle both water and fertilizer solutions, dispensing them as needed based on plant growth stages.

4.Solar Power Option

Integrating solar panels would make the Smart Flower Pot more energy-efficient and eco-friendly. This option would allow the system to run independently without relying on external power sources, ideal for users who want a sustainable solution.

5.Voice Assistant Compatibility

Integrating with voice assistants like Alexa, Google Assistant, or Siri would allow users to interact with the flower pot through voice commands, enhancing convenience and accessibility.

7. References.

1) https://youtu.be/gjYcLOoiH6c?si=qR-hiqdwRjjNfNuj
2) *https://github.com/intellar/oled_eye_display*
3) *https://youtube.com/shorts/SV6P0KDAPRM?si=xrY8fo3OCji_s5oM*
4) *https://youtu.be/fNg8nQjqIq4?si=19D9Rv6bOHlcwQge*