Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Digital Assignment – 1

My Contribution in J-Component(Project)

Name: A.Ramesh kumaran

Reg.No.: 22MIC0072

Course name: Augmented Reality and Virtual Reality

Course code: CSI4005

Submitted to: Dr. Karthik K

My Contribution : [Hardware IoT Setup, Cloud integration, OpenCV implementation and Multimodal interaction] for "Smart and Interactive Gardening Assistant using AR and IoT" project.

-A.Ramesh kumaran
22MIC0072

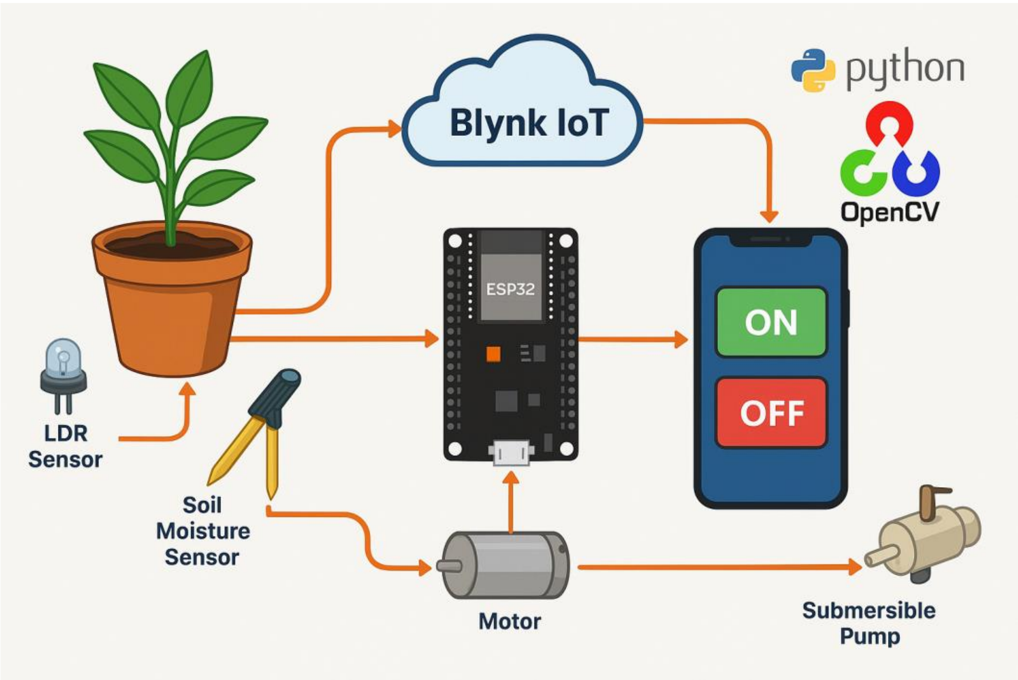
Project Overview:

This novel project is a "Smart and Interactive Gardening Assistant" that uses the combination of Augmented Reality (AR) and IoT technologies to create an engaging and educational plant care experience. It integrates multimodal interaction including hand gestures, feedback and speech recognition for intuitive control of devices like LEDs, submersible water pumps, soil moisture sensors, and LDRs.

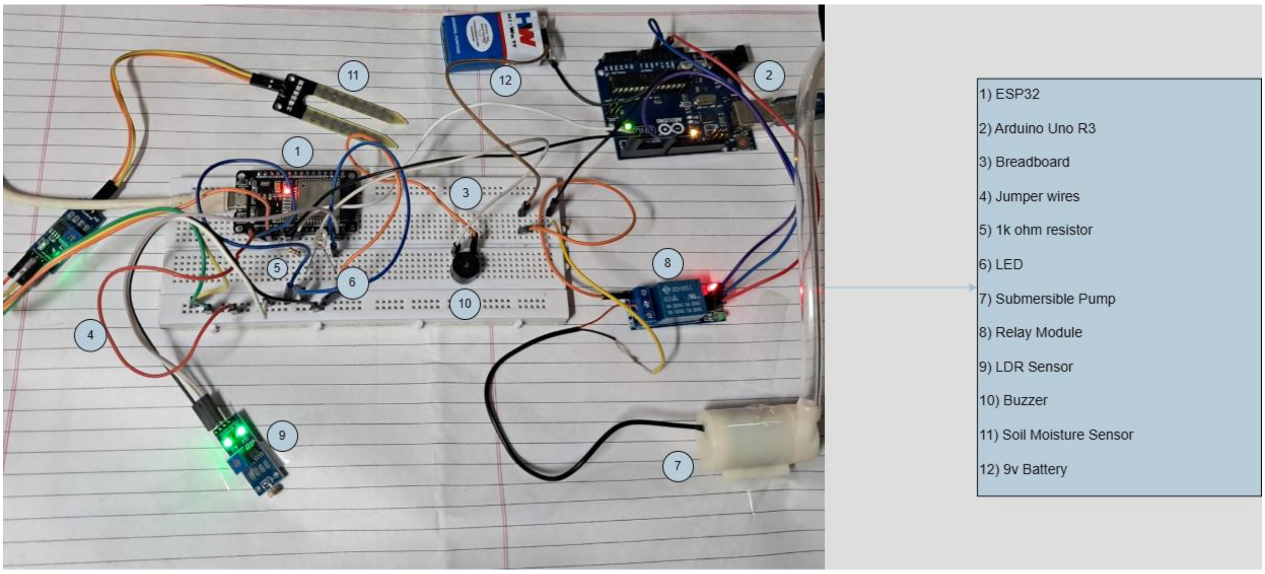
Users can analyze real-time environmental data such as soil moisture and light intensity, and control watering or lighting through virtual buttons in a mobile OpenCV + Python-based AR app. The system is especially ideal for kids who want to explore gardening without getting their hands dirty, while also learning the basics of plant health monitoring and automation.

The smart assistant is also accessible remotely from anywhere in the world via Blynk IoT, enabling users to monitor and manage plant care through their smartphone—whether at home or away.

System Architecture:



Hardware Setup and Firmware Development:



1) ESP32

- **Purpose:** Acts as the Wi-Fi-enabled controller.
- **Functionality:**
    - Connects to Blynk IoT via Wi-Fi to receive commands (like turning ON/OFF pump) remotely.
    - Reads data from sensors like the Soil Moisture Sensor and LDR.
    - Sends "ON/OFF" signals to the Arduino via Serial Communication.

2) Arduino Uno R3

- **Purpose:** Acts as the motor controller.
- **Functionality:**
    - Receives ON/OFF commands from ESP32 through Serial Communication.
    - Controls the relay module, which in turn powers the submersible pump.

3) Breadboard

- **Purpose:** A prototyping platform to connect components without soldering.
- **Functionality:** Hosts connections between ESP32, resistors, LED, buzzer, and sensors.

4) Jumper Wires

- **Purpose:** Used to interconnect the ESP32, sensors, relay, and Arduino on the breadboard.

- **Functionality:** Ensures data and power flow between all devices.

5) 1k ohm Resistor

- **Purpose:** Current-limiting resistor.
- **Functionality:** Protects the LED (component 6) from drawing too much current and burning out.

6) LED

- **Purpose:** Status indicator (e.g., relay/motor ON/OFF).
- **Functionality:** Lights up based on Blynk input; can be used to verify logic signal from ESP32 to Arduino.

7) Submersible Pump

- **Purpose:** Waters the plant.
- **Functionality:** Turned ON/OFF by the relay (8) controlled by Arduino, based on the ESP32's command.

8) Relay Module

- **Purpose:** Electrically isolates and controls high-power devices (like the pump).
- **Functionality:** When the relay receives a signal from Arduino, it switches the 9V battery supply to the pump ON or OFF.

9) LDR Sensor

- **Purpose:** Detects ambient light level.
- **Functionality:** Sends digital input to ESP32. When it's dark, ESP32 turns ON the buzzer (10) as a signal.

10) Buzzer

- **Purpose:** Audible alert.
- **Functionality:** Beeps when the LDR detects darkness (could indicate night, so don't water then).

11) Soil Moisture Sensor

- **Purpose:** Measures moisture level in soil.
- **Functionality:** Analog signal to ESP32. If moisture is low, the system suggests watering the plant.

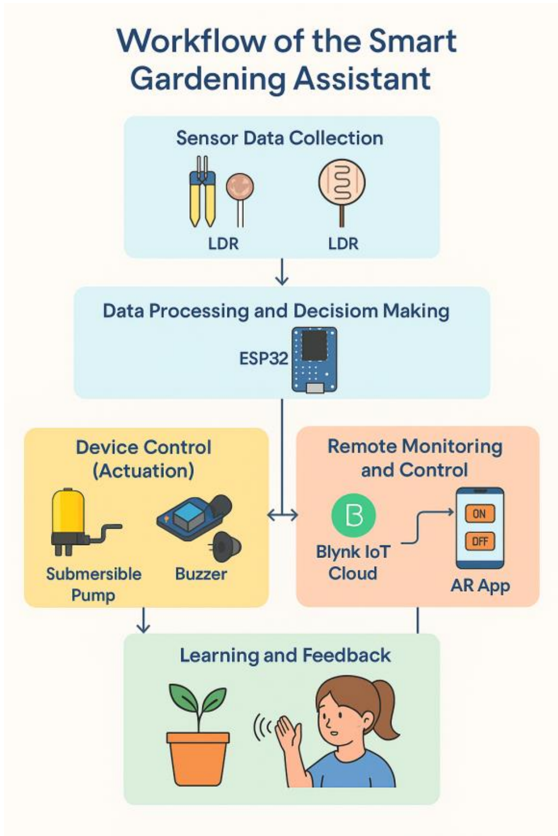
12) 9V Battery

- **Purpose:** Provides external power to the submersible pump via the relay.
- **Functionality:** Pump requires more current than ESP32/Arduino can supply directly, so an external battery is essential.

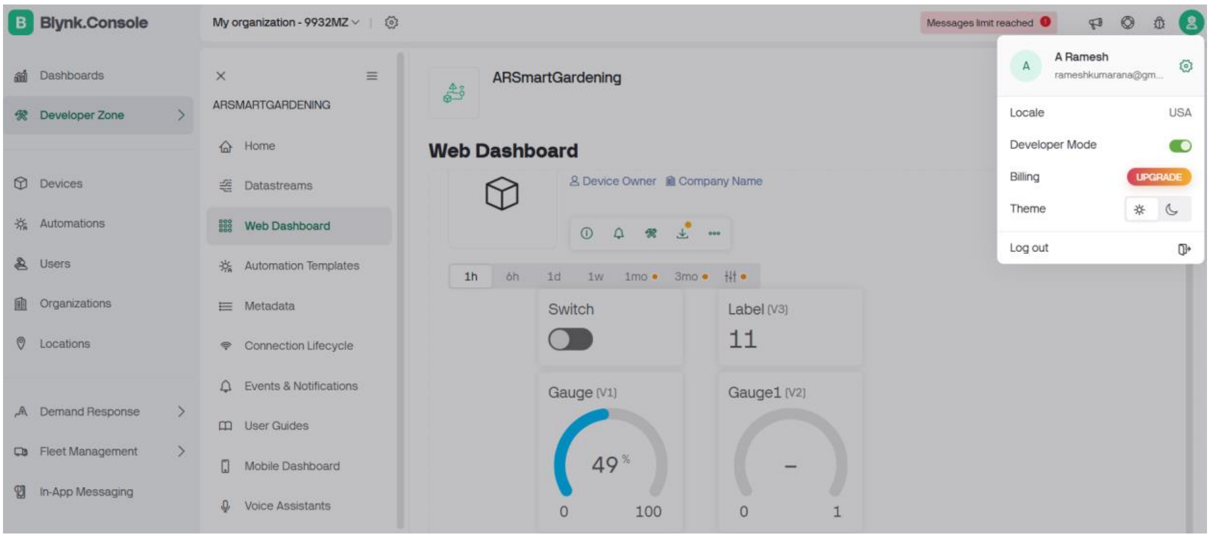
System workflow:

1) **Sensor Monitoring**
Soil moisture and LDR sensors continuously collect real-time data about plant health and surroundings.
2) **Data Transmission**
Sensor data is sent to the ESP32, which processes and uploads it to the Blynk IoT Cloud.
3) **User Interaction**
Users interact via the AR mobile app with:
- Virtual Buttons (OpenCV)
- Hand Gestures
- Voice Commands
- Feedback

4) **Remote Control**
Commands from the mobile app (e.g., water plant) are sent to ESP32, which communicates with the Arduino via Serial.
5) **Action Execution**
Arduino activates connected devices like the Submersible Pump, Buzzer, or LED based on received commands.
6) **Smart Gardening**
The system autonomously or manually cares for plants from remote location, ensuring engagement and learning for kids—without getting their hands dirty!



Create a Template in Blynk IoT to configure the virtual buttons for the led, submersible pump, LDR sensor, and soil moisture sensor to receive the data and display the data in the app



ESP32 Code in Arduino IDE:

```
// #define BLYNK_TEMPLATE_ID "TMPL3rLB1CbSW"
// #define BLYNK_TEMPLATE_NAME "Led Blink"
// #define BLYNK_AUTH_TOKEN "kGWvdkB1V91FWflMiw24KFLFztnmKPZD"
#define BLYNK_TEMPLATE_ID "TMPL3zlJravNQ"
#define BLYNK_TEMPLATE_NAME "ARGardenAssistant"
#define BLYNK_AUTH_TOKEN "oqVvTjuTCSu7gj4_mOiyvl1ToIfarOtb"

#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <BlynkSimpleEsp32.h>

char ssid[] = "OpenWifi";
char pass[] = "hackers12345";

#define LED_PIN 2
#define SOIL_SENSOR_PIN 15
#define LDR_SENSOR_PIN 22
#define BUZZER_PIN 21

void setup()
{
  Serial.begin(115200);
  Serial2.begin(9600, SERIAL_8N1, 16, 17);

  pinMode(LED_PIN, OUTPUT);
  pinMode(SOIL_SENSOR_PIN, INPUT);
  pinMode(LDR_SENSOR_PIN, INPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);

  delay(1000);
}


void loop()
{
  Blynk.run();
  sendSensorData();
}

BLYNK_WRITE(V0)
{
  int pinValue = param.asInt();
  digitalWrite(LED_PIN, pinValue ? HIGH : LOW);
  if (pinValue) {
    Serial2.println("ON");
  } else {
    Serial2.println("OFF");
  }
}

void sendSensorData()
{
```

```
    int ldrValue = digitalRead(LDR_SENSOR_PIN);

    int moistureValue = analogRead(SOIL_SENSOR_PIN);
    int moisturePercentage = map(moistureValue, 0, 4095, 100, 0);
    Blynk.virtualWrite(V1, moistureValue);
    Blynk.virtualWrite(V3, moisturePercentage);


    Blynk.virtualWrite(V2, ldrValue);

    Serial.print("Moisture: ");
    Serial.print(moisturePercentage);
    Serial.print("% | LDR: ");
    Serial.println(ldrValue);

    if (ldrValue ==1)
    {
      digitalWrite(BUZZER_PIN, HIGH);
      Serial.println("Buzzer ON (Dark Environment)");
    }
    else
    {
      digitalWrite(BUZZER_PIN, LOW);
    }

    delay(2000); // Update every 2 seconds
}
```

Python Code for OpenCV Hand Gestures, Virtual buttons and Multimodal interactions:

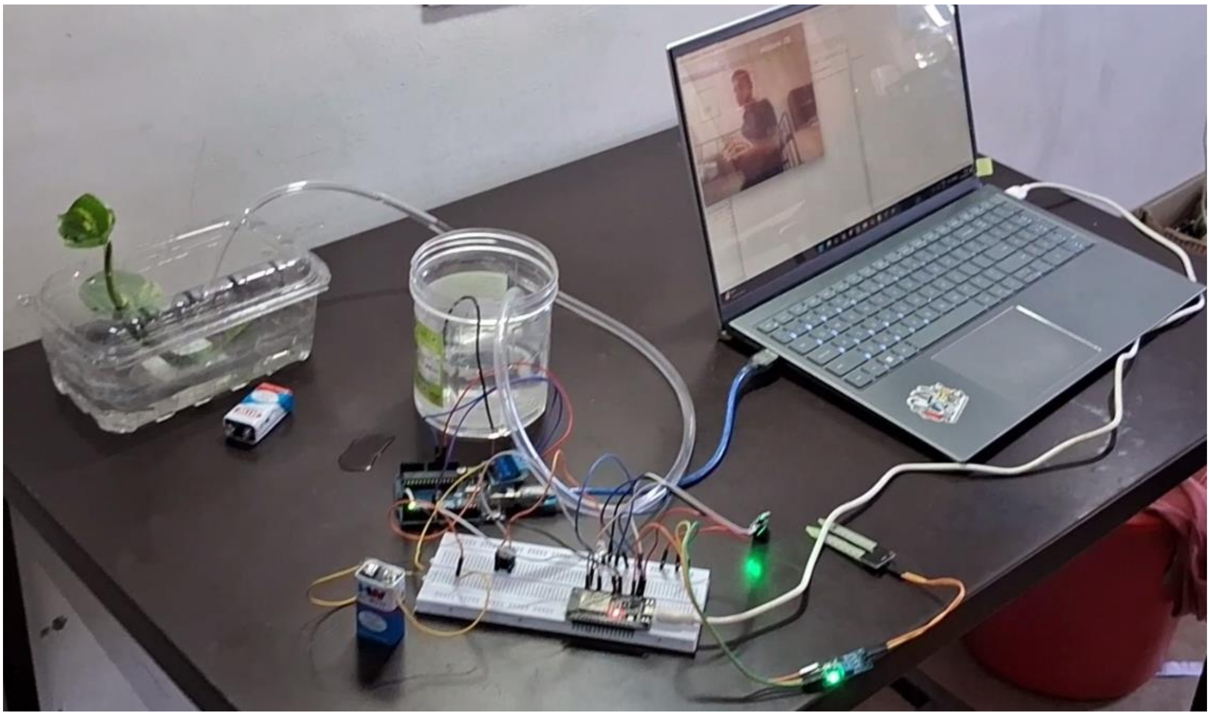
```
import cv2
import mediapipe as mp
import requests
import speech_recognition as sr
import threading
import time

# Blynk API URLs
BLYNK_AUTH = "oqVvTjuTCSu7gj4_mOiyvl1ToIfarOtb"  # Replace with your Blynk token
LED_ON_URL = f"https://blynk.cloud/external/api/update?token={BLYNK_AUTH}&v0=1"
LED_OFF_URL = f"https://blynk.cloud/external/api/update?token={BLYNK_AUTH}&v0=0"
MOISTURE_URL = f"https://blynk.cloud/external/api/get?token={BLYNK_AUTH}&v1"  #
Virtual pin V1 for moisture
LDR_URL = f"https://blynk.cloud/external/api/get?token={BLYNK_AUTH}&v2"  # Virtual pin
V2 for LDR

# Initialize MediaPipe Hand tracking
mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
hands = mp_hands.Hands(min_detection_confidence=0.5, min_tracking_confidence=0.5)
```

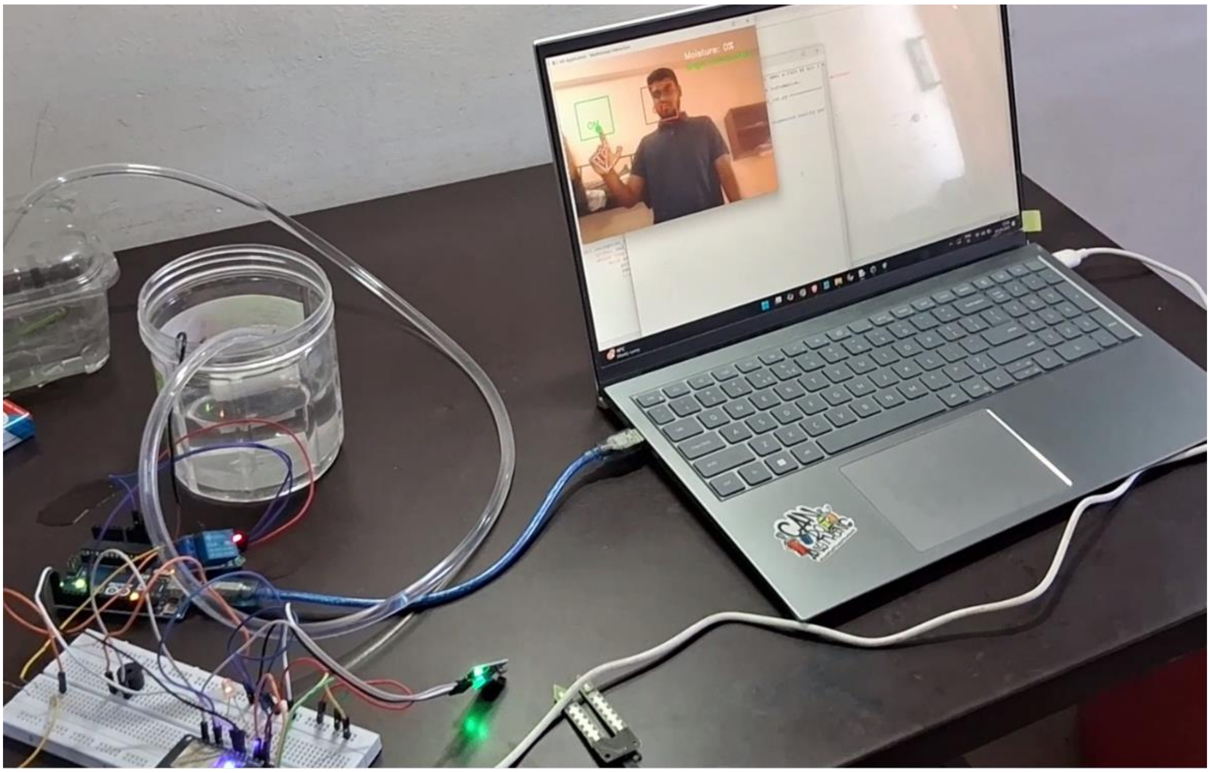
```
# Open webcam
cap = cv2.VideoCapture(0)
led_on = False

# Initialize Speech Recognizer
recognizer = sr.Recognizer()

# Set FPS limit
FPS = 10  # Desired FPS
frame_time = 1.0 / FPS

def recognize_speech():
    global led_on
    while True:
        with sr.Microphone() as source:
            print("Listening for voice commands...")
            recognizer.adjust_for_ambient_noise(source, duration=1)
            try:
                audio = recognizer.listen(source, timeout=10, phrase_time_limit=5)
                command = recognizer.recognize_google(audio).lower()
                print(f"Recognized: {command}")

                if "turn on" in command and not led_on:
                    print("Turning LED ON via Blynk (Voice)")
                    requests.get(LED_ON_URL)
                    led_on = True
                elif "turn off" in command and led_on:
                    print("Turning LED OFF via Blynk (Voice)")
                    requests.get(LED_OFF_URL)
                    led_on = False
            except sr.WaitTimeoutError:
                print("Listening timed out. No speech detected, retrying...")
            except sr.UnknownValueError:
                print("Could not understand the command")
            except sr.RequestError:
                print("Could not request results, check your internet connection")

# Run speech recognition in a separate thread
speech_thread = threading.Thread(target=recognize_speech, daemon=True)
speech_thread.start()

while cap.isOpened():
    start_time = time.time()

    ret, frame = cap.read()
    if not ret:
        continue

    frame = cv2.flip(frame, 1)
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(rgb_frame)

    # Fetch soil moisture data from Blynk
    try:
```

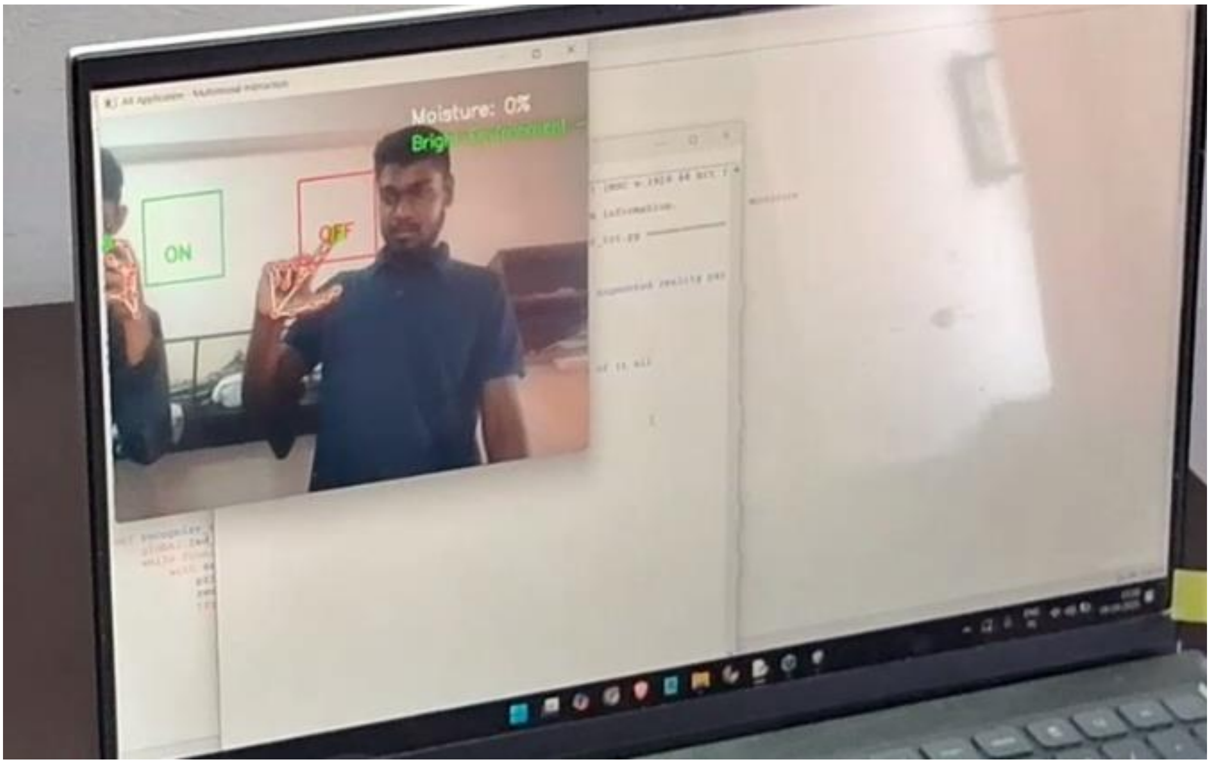
```
        response = requests.get(MOISTURE_URL)
        moisture_level = response.text if response.status_code == 200 else "N/A"
    except:
        moisture_level = "Error"

    # Fetch LDR sensor data from Blynk
    try:
        response = requests.get(LDR_URL)
        ldr_value = response.text if response.status_code == 200 else "N/A"
    except:
        ldr_value = "Error"

    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(frame, hand_landmarks,
mp_hands.HAND_CONNECTIONS)
            index_finger_tip =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP]
            x, y = int(index_finger_tip.x * frame.shape[1]), int(index_finger_tip.y * frame.shape[0])
            cv2.circle(frame, (x, y), 10, (0, 255, 0), -1)

            # Define ON/OFF button areas
            on_area = (50, 100, 150, 200)
            off_area = (250, 100, 350, 200)

            if on_area[0] < x < on_area[2] and on_area[1] < y < off_area[3] and not led_on:
                print("Turning LED ON via Blynk (Gesture)")
                requests.get(LED_ON_URL)
                led_on = True
            elif off_area[0] < x < off_area[2] and off_area[1] < y < off_area[3] and led_on:
                print("Turning LED OFF via Blynk (Gesture)")
                requests.get(LED_OFF_URL)
                led_on = False

    # Draw ON/OFF button areas
    cv2.rectangle(frame, (50, 100), (150, 200), (0, 255, 0), 2)
    cv2.putText(frame, "ON", (75, 175), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)
    cv2.rectangle(frame, (250, 100), (350, 200), (0, 0, 255), 2)
    cv2.putText(frame, "OFF", (275, 175), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 2)

    # Display soil moisture reading
    cv2.putText(frame, f"Moisture: {moisture_level}%", (400, 50),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)

    # Display LDR sensor reading and environment status
    if ldr_value == "0":
        ldr_text = "Bright Environment - Sufficient Sunlight"
        ldr_color = (0, 255, 0)  # Green
    elif ldr_value == "1":
        ldr_text = "Dark Environment - Needs Sunlight"
        ldr_color = (0, 0, 255)  # Red
    else:
        ldr_text = "LDR Sensor Error"
        ldr_color = (0, 255, 255)  # Yellow
```

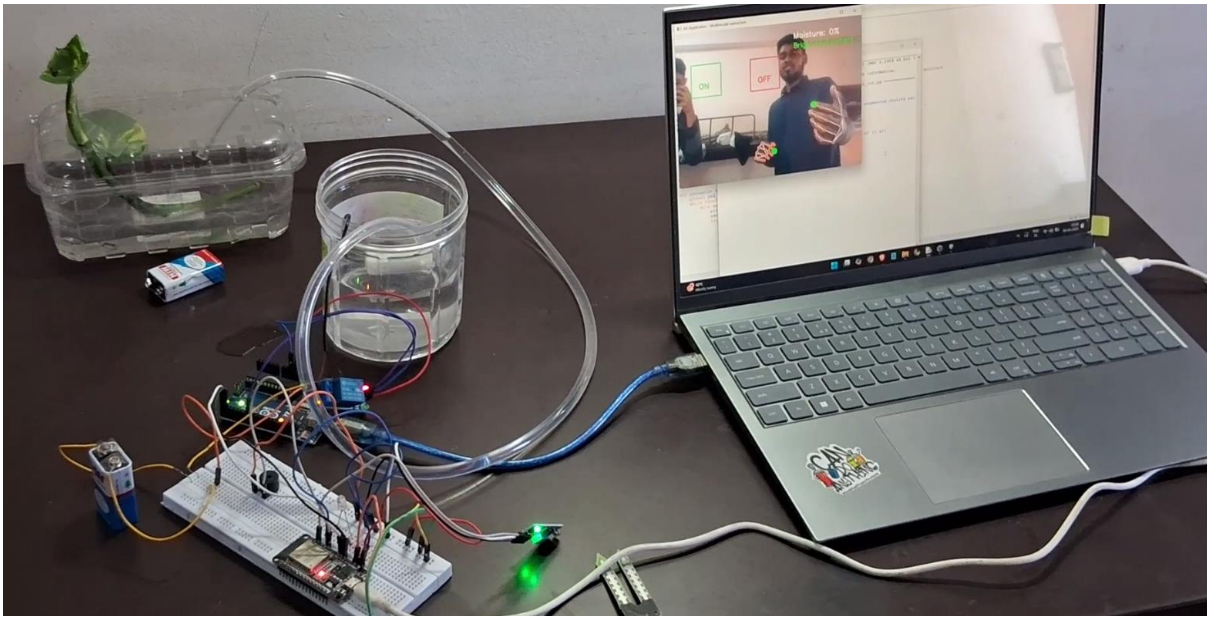
```
    cv2.putText(frame, ldr_text, (400, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.7, ldr_color, 2)

    cv2.imshow("AR Application - Multimodal interaction", frame)

    # FPS control
    elapsed_time = time.time() - start_time
    if elapsed_time < frame_time:
        time.sleep(frame_time - elapsed_time)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Practical Demonstration:




Moisture: 0%
ON


Moisture: 0%
ON
OFF



GitHub Link of the project:

https://github.com/ARamesh-tech/GreenTalk-AR-Powered-Smart-Gardening-Assistant-Utility-Control

Demonstration video link:

https://drive.google.com/file/d/1KjkynaLgwyfoLt8SuXTiraEqS0ezpdiR/view?usp=sharing

****