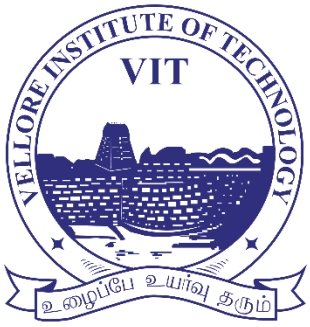


VIT®

Vellore Institute of Technology

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

Project Report – J Component

Team Members:

Name	Registration Number
A. Ramesh Kumaran	22MIC0072
Utkarsh Kumar	22MIC0087
Arpit Dixit	22MIC0028
Somasundaram T	22MID0018

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

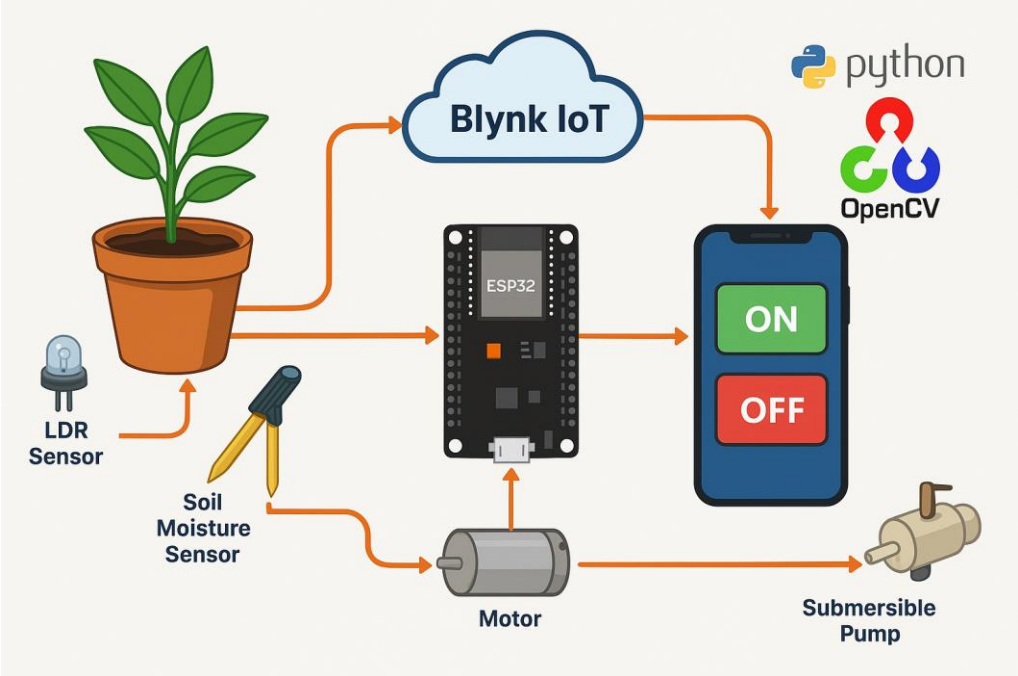
Project Overview:

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.

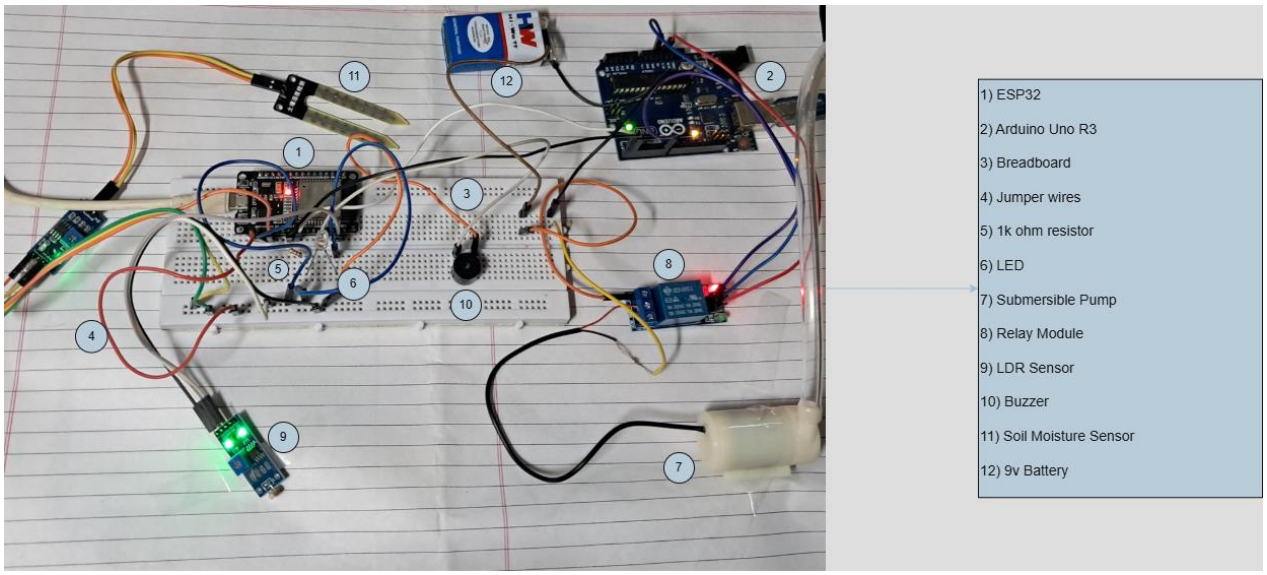
Users can analyze real-time environmental data such as soil moisture and light intensity, and control watering or lighting through virtual buttons in a mobile OpenCV + Python-based AR app. The system is especially ideal for kids who want to explore gardening without getting their hands dirty, while also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

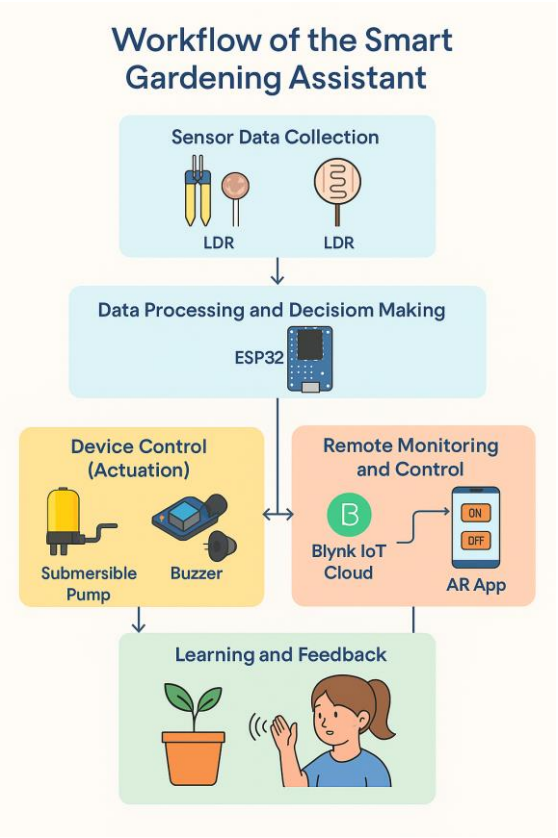
- **Purpose:** Provides external power to the submersible pump via the relay.
- **Functionality:** Pump requires more current than ESP32/Arduino can supply directly, so an external battery is essential.

System workflow:

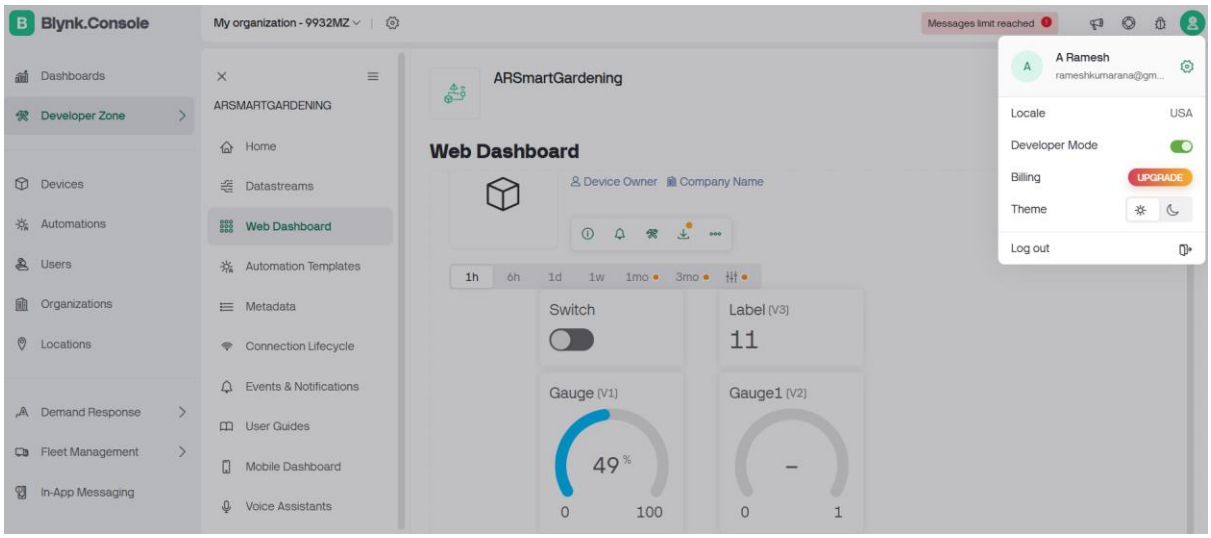
1) **Sensor Monitoring**
Soil moisture and LDR sensors continuously collect real-time data about plant health and surroundings.
2) **Data Transmission**
Sensor data is sent to the ESP32, which processes and uploads it to the Blynk IoT Cloud.
3) **User Interaction**
Users interact via the AR mobile app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

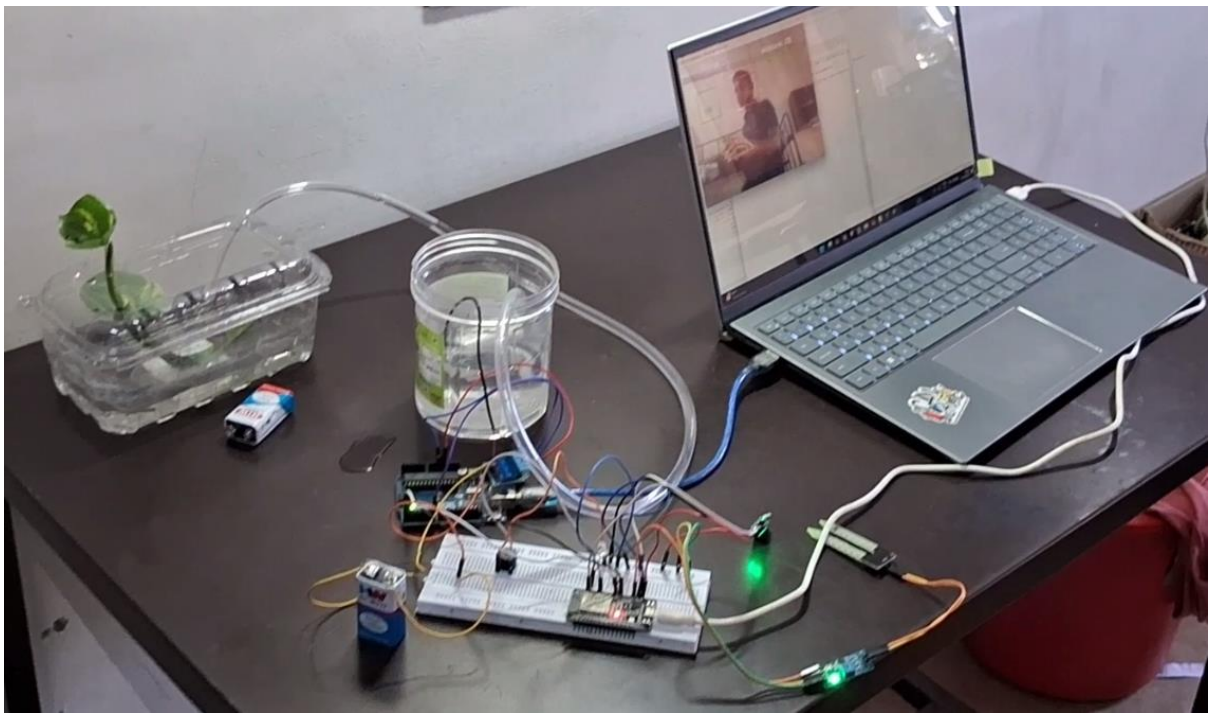
```
import cv2
import mediapipe as mp
import requests
import speech_recognition as sr
import threading
import time

# 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)
```

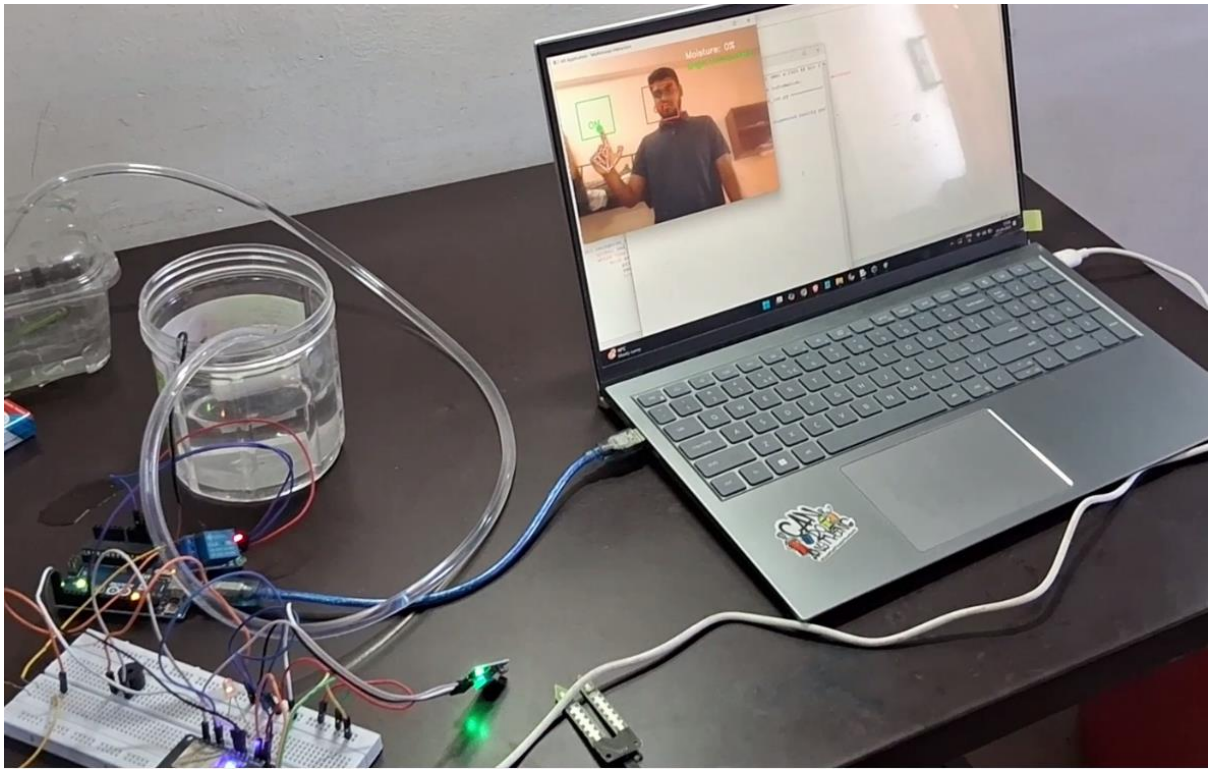
```
# Open webcam
cap = cv2.VideoCapture(0)
led_on = False

# 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:
```

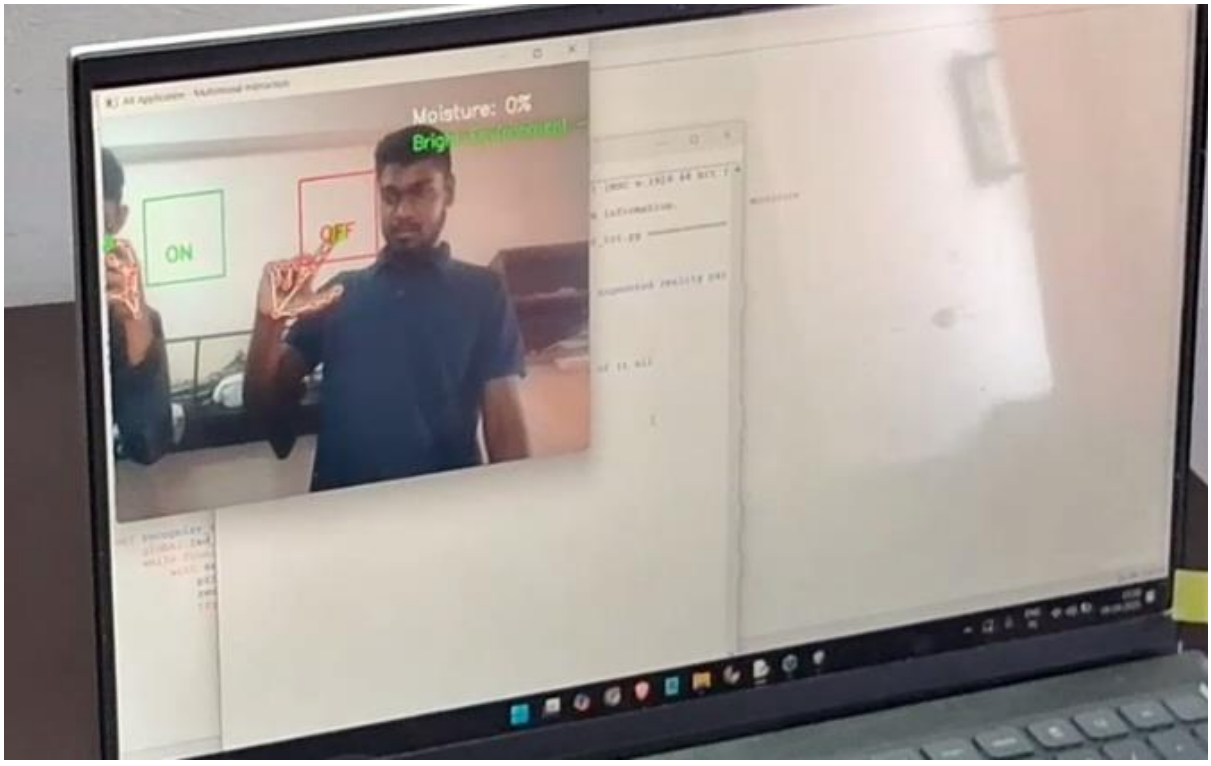
```
        response = requests.get(MOISTURE_URL)
        moisture_level = response.text if response.status_code == 200 else "N/A"
    except:
        moisture_level = "Error"

    # 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
```

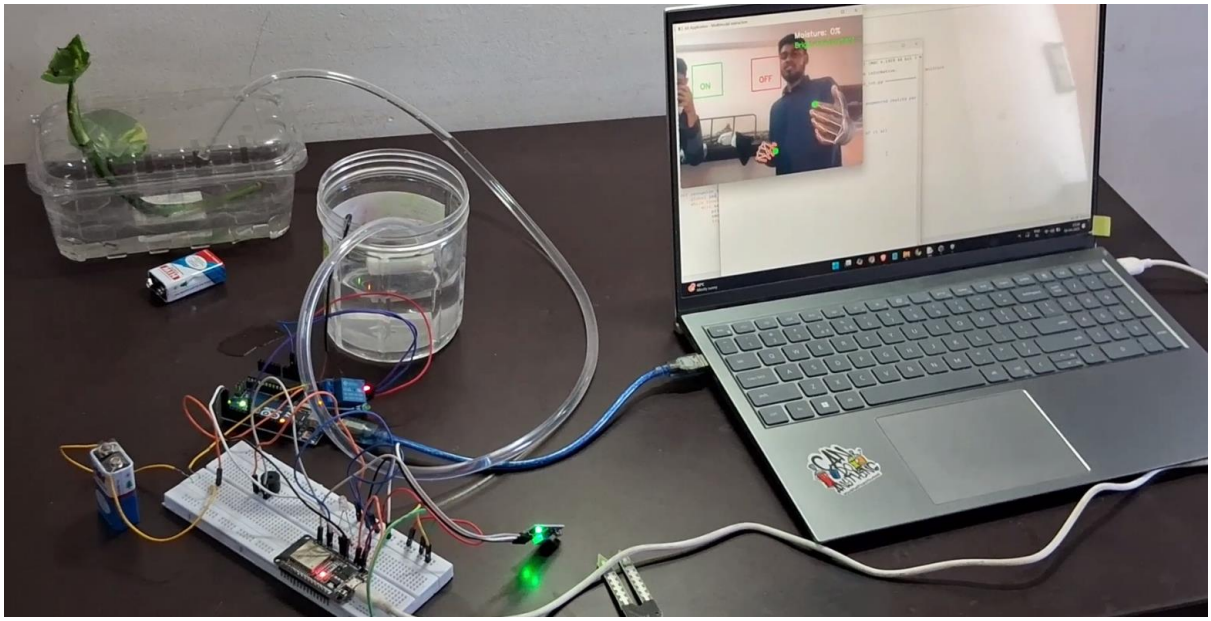
```
    cv2.putText(frame, ldr_text, (400, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.7, ldr_color, 2)

    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
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

****

Contribution Report: Smart and Interactive Gardening Assistant using AR and IoT

-Utkarsh Kumar 22MIC0087

Project Overview

The "Smart and Interactive Gardening Assistant using AR and IoT" is a novel project combining Augmented Reality (AR) with Internet of Things (IoT) technologies to revolutionize modern gardening. This assistant enables real-time monitoring, automation, and immersive interactions with plants, using sensor data and AR visualizations. The project is designed to help users— especially urban gardeners and plant enthusiasts—care for their plants more efficiently and engagingly.

My contribution to this project focused on two major areas:

1. Integration of the Blynk IoT API for real-time monitoring and control.
2. Development of a marker-based AR feature that recognizes a notepad image and generates a 3D plant model in response.

1. Blynk IoT API Integration

This component was key to enabling remote control and monitoring of the garden through a smartphone interface.

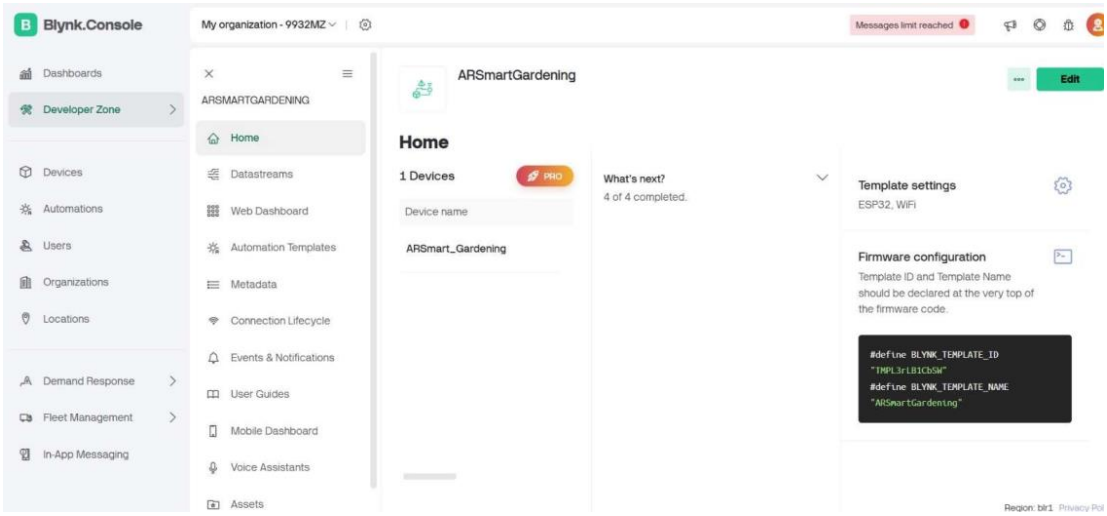
Steps Involved:

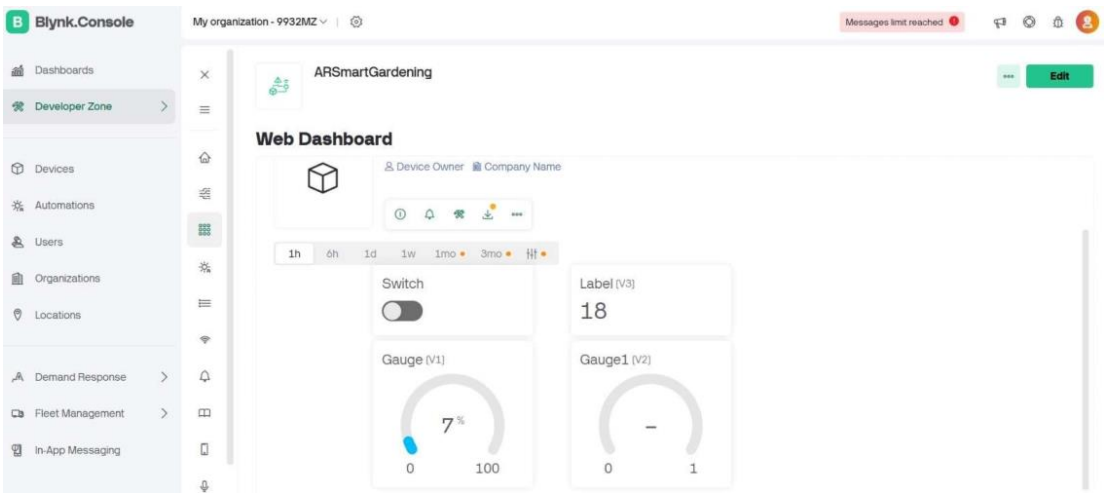
A. **Blynk Project Configuration**

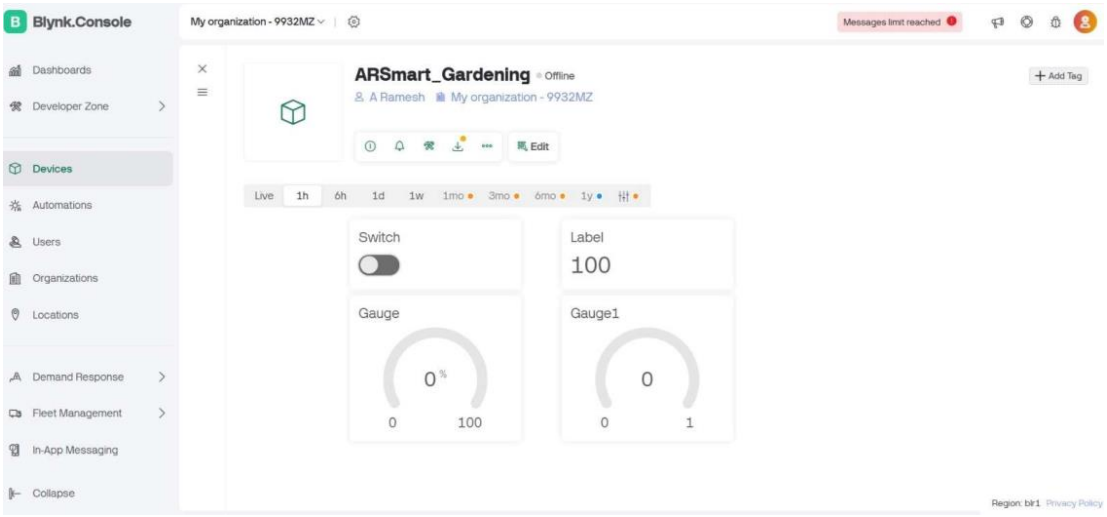
- I created a Blynk IoT app project using the ESP8266 board template.
- An auth token was generated and integrated into the ESP8266 firmware.
- The app interface was built using:
  - o Gauges to show sensor readings o Switches to control the pump manually o Notification alerts for low moisture or high temperature

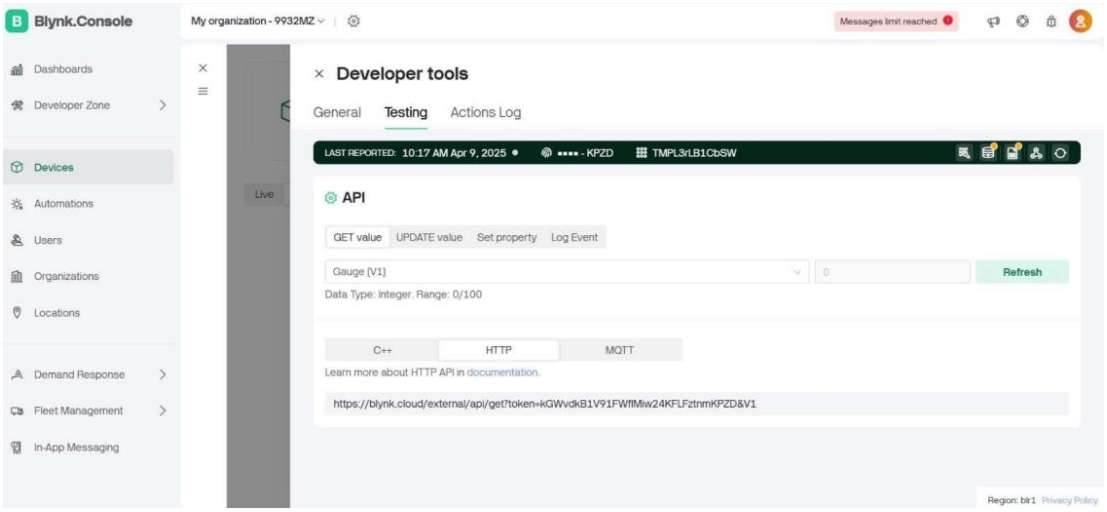
B. **Testing and Optimization**

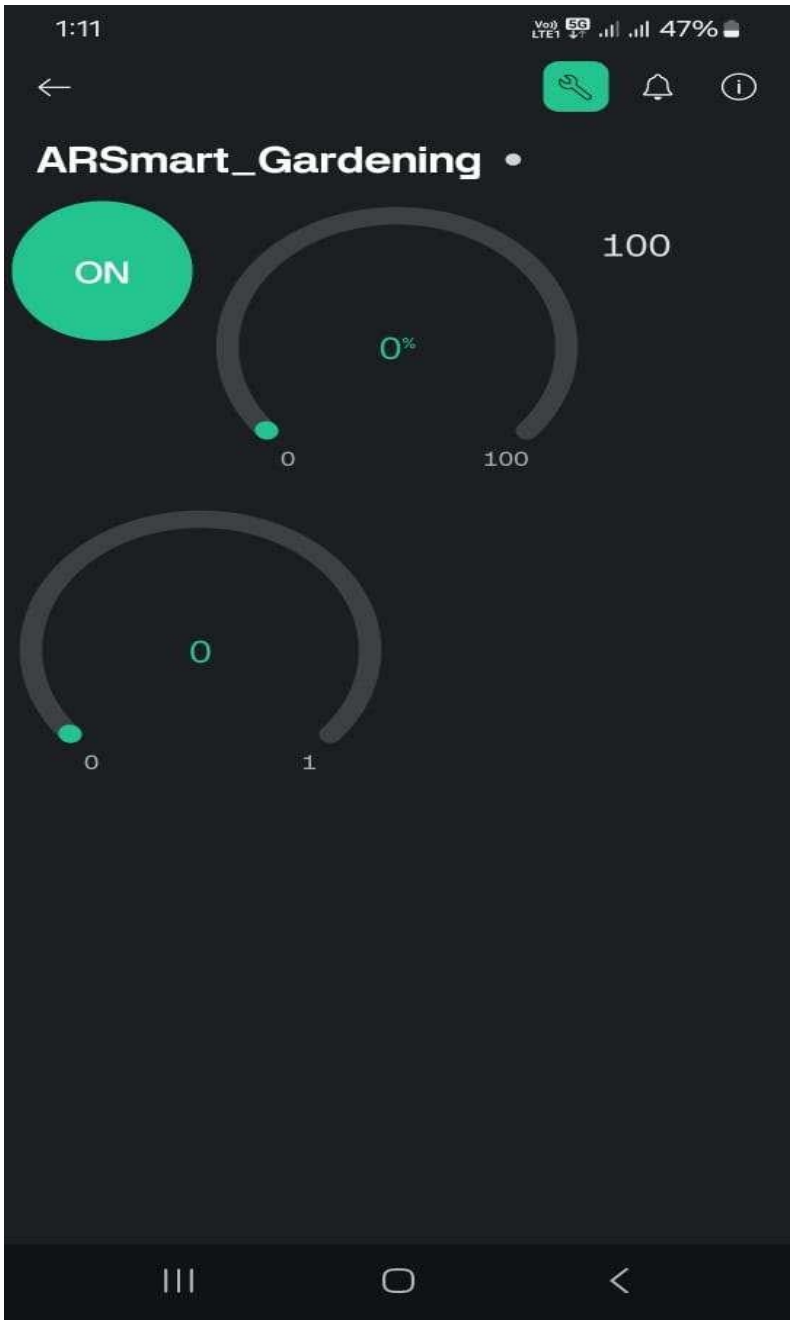
- I rigorously tested the system for real-time updates, responsiveness, and accuracy.
- Fine-tuning was done for sensor calibration and pump timings to optimize plant watering.


Blynk.Console
My organization - 9932MZ
Messages limit reached
Dashboards
Developer Zone
Devices
Automations
Users
Organizations
Locations
Demand Response
Fleet Management
In-App Messaging
ARSMARTGARDENING
Home
Datastreams
Web Dashboard
Automation Templates
Metadata
Connection Lifecycle
Events & Notifications
User Guides
Mobile Dashboard
Voice Assistants
Assets
ARSmartGardening
Edit
Home
1 Devices
PRO
Device name
ARSmart_Gardening
What's next?
4 of 4 completed.
Template settings
ESP32, WiFi
Firmware configuration
Template ID and Template Name should be declared at the very top of the firmware code.
#define BLYNK_TEMPLATE_ID
"TMPL3rLB1CbSW"
#define BLYNK_TEMPLATE_NAME
"ARSmartGardening"
Region: blr1


Blynk.Console
My organization - 9932MZ
Messages limit reached
Dashboards
Developer Zone
Devices
Automations
Users
Organizations
Locations
Demand Response
Fleet Management
In-App Messaging
ARSmartGardening
Edit
Web Dashboard
Device Owner
Company Name
1h
6h
1d
1w
1mo
3mo
Switch
Label (V3)
18
Gauge (V1)
7 %
0
100
Gauge1 (V2)
–
0
1


Blynk.Console
My organization - 9932MZ
Messages limit reached
Dashboards
Developer Zone
Devices
Automations
Users
Organizations
Locations
Demand Response
Fleet Management
In-App Messaging
Collapse
ARSmart_Gardening
Offline
A Ramesh
My organization - 9932MZ
Add Tag
Edit
Live
1h
6h
1d
1w
1mo
3mo
6mo
1y
Switch
Label
100
Gauge
0 %
0
100
Gauge1
0
0
1
Region: blr1
Privacy Policy


Blynk.Console
My organization - 9932MZ
Messages limit reached
Dashboards
Developer Zone
Devices
Automations
Users
Organizations
Locations
Demand Response
Fleet Management
In-App Messaging
Developer tools
General
Testing
Actions Log
LAST REPORTED: 10:17 AM Apr 9, 2025
TMPL3rLB1CbSW
Live
API
GET value
UPDATE value
Set property
Log Event
Gauge (V1)
Refresh
Data Type: Integer. Range: 0/100
C++
HTTP
MQTT
Learn more about HTTP API in documentation.
https://blynk.cloud/external/api/get?token=kGWvdkB1V91FWfIMiw24KFLFztnmKPZD&V1
Region: blr1
Privacy Policy


1:11
47%
ARSmart_Gardening
ON
100
0%
0
100
0
0
1

2. **Marker-Based AR Implementation**

To enhance user interaction, I developed a marker-based AR system that detects a specific image of a notepad (which can be later changed to say pot) and spawns a virtual plant model in its place. This was a crucial part of the AR module of the project, linking the digital world to the physical environment in a visually compelling way.

Steps Involved:

a. **Marker Design**

- A high-contrast notepad image was selected as the AR marker.
- It was tested to ensure consistent detection under varied lighting conditions.

b. **AR Development Environment**

- I used Unity 3D along with the Vuforia Engine for marker-based AR development.
- The Vuforia SDK allowed me to:
  - Upload the notepad image as a marker (Image Target) o Attach a

  3D model of a plant to be rendered when the marker is detected c.

  Model Integration

- I imported a low-poly 3D plant model into Unity.
- The model was positioned and scaled appropriately to appear naturally over the notepad marker.
- Added a rotation animation to make the AR experience more engaging.

  •Quick and stable marker recognition

  • Realistic rendering of the plant

  •Smooth camera tracking

Impact of My Contribution

- The Blynk integration enabled real-time, user-friendly interaction with garden hardware through mobile phones, providing automated plant care and insights.
- The AR component o ered a futuristic, immersive experience, bridging the physical and virtual worlds by letting users see a digital plant appear magically over a simple notepad.

Together, these contributions were instrumental in fulfilling the project's vision of a truly Smart and Interactive Gardening Assistant. They demonstrated the powerful synergy of IoT and AR, enhancing both the utility and appeal of gardening in modern times.

NAME: T. SOMASUNDARAM

REGNO: 22MID0018

J-COMPONENT TOPIC

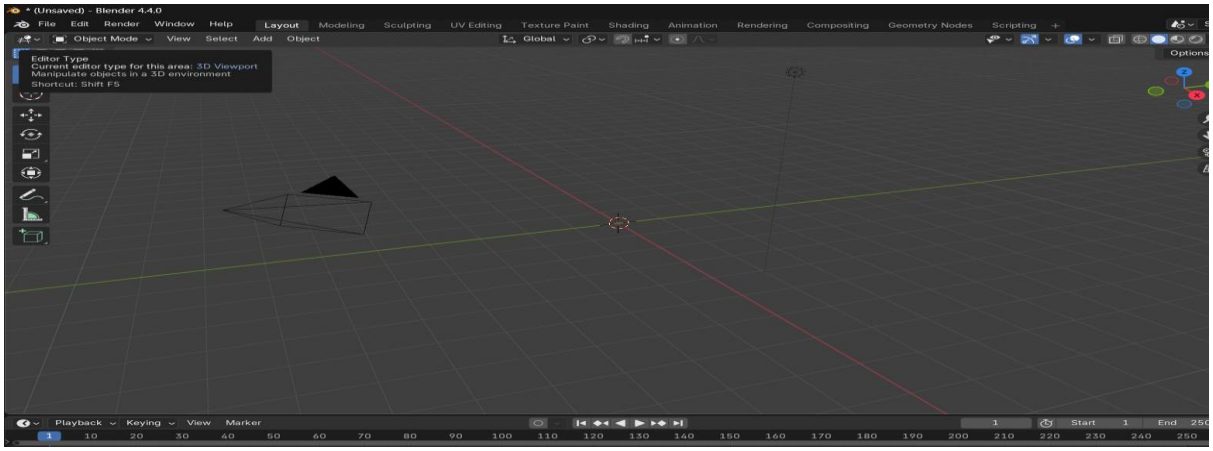
GREENTALK: AR POWERED SMART & INTERACTIVE GARDENING ASSISTANT

MY CONTRIBUTION: CREATING 3D OBJECT ON BLENDER

BLENDER APP

Blender is a free and open-source 3D creation software. It's used for:

- **3D modeling**
- **Animation**
- **Texturing & materials**
- **Rendering (realistic images)**
- **VFX (visual effects)**
- **Game assets**
- **Motion graphics**
- **Video editing**
- **Sculpting, and more!**



Basic of blender app

Step 1: Open Blender

When you first open Blender, you'll see the default cube in the center of the scene.

You can:

- Use it as a base for modeling.
- Or delete it:
    - Click the cube to select it.
    - Press X or Delete, then confirm.

Step 2: Add a Basic Object (Primitive)

Blender offers a variety of primitive shapes you can use as a foundation.

How to Add:

1. Press Shift + A to open the Add menu.
2. Go to Mesh.
3. Choose from:
    - Cube – Standard box
    - UV Sphere – Perfect for balls/planets
    - Cylinder – Great for tubes, wheels
    - Cone – Useful for spikes, tips

- Torus – Donut shape, rings
- Plane – Flat surface, great for floors
- Monkey (Suzanne) – A fun, built-in test object

Step 3: Enter Edit Mode to Customize the Shape

Switching Modes:

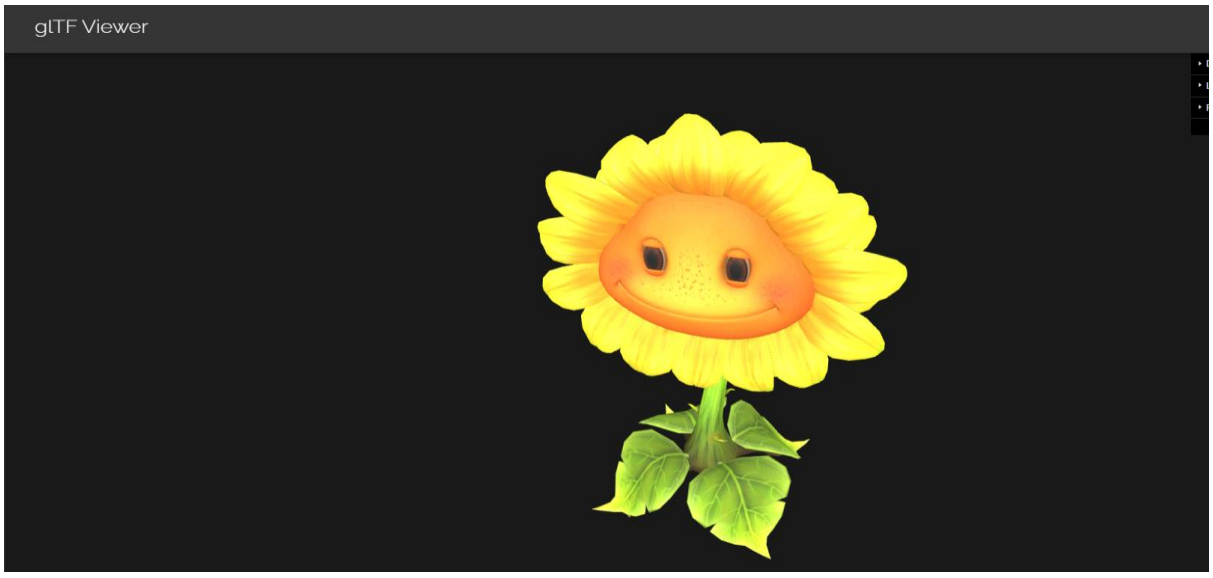
- Select your object.
- Press Tab → enter Edit Mode (or choose from top-left dropdown).

You can switch between:

- Vertex select (point)
- Edge select (line)
- Face select (square)
  Use the buttons at the top-left of the viewport or 1, 2, 3 keys.

This is where creativity shines! Use combinations of tools to create complex shapes:

Creating 3d object



Step 1: Set Up Your Scene

1. Open **Blender**.
2. Delete the default cube (X → Delete).

Step 2: Create the Center (Sunflower Seed Disk)

1. Press Shift + A → Mesh → UV Sphere.
2. Press S to **scale** it down (e.g., S → 0.5).
3. Go to the **Modifiers tab** → Add **Subdivision Surface** modifier for smoothness.
4. Right-click the sphere → **Shade Smooth**.
5. This is your **seed center**.

Step 3: Create a Petal

1. Shift + A → Mesh → Plane
2. Enter **Edit Mode** (Tab)
3. Select all → S Y 0.2 → to make it narrow.
4. S X 2 → stretch it to look like a petal.
5. Add a **Subdivision Surface** modifier for smooth curves.
6. Right-click → Shade Smooth.
7. In Object Mode, rotate and position it to sit slightly beside the sunflower center.
8. Add **solidify modifier** if it's too thin.

Step 4: Duplicate Petals Around the Center

1. With the petal selected, go to the **Modifiers tab**.
2. Add a **Array Modifier** → set Count to 1.
3. Then add a **Simple Deform Modifier**:
    - Set to **Bend**.
    - Set **Angle** to 360°.
    - Set **Origin** to a new empty at the center of the flower:
        - Shift + A → Empty → Plain Axes
        - Select the petal → Set the Empty as its origin in modifier.

4. Or use this faster trick:
    - Press Shift + D to duplicate the petal.
    - Press R Z (angle) → rotate it around the seed disk.
    - Repeat until you form a full flower (usually ~20–30 petals).

Step 5: Add the Stem

1. Shift + A → Mesh → Cylinder
2. In the options (bottom left), set vertices to 16 and make it long/thin.
3. Scale it (S) to a narrow vertical shape.
4. Position it below the seed center.

Step 6: (Optional) Add Leaves

1. Duplicate a petal → Scale it larger and flatten it.
2. Move it to the stem area and rotate as a leaf.
3. Add veins with the knife tool or texture later.

Step 7: Add Materials

1. Select parts of the flower:
    - Petals → Yellow
    - Center → Brown or Black
    - Stem → Green
2. Go to **Material Properties** tab (red sphere icon) → Click **New** → Choose base color.

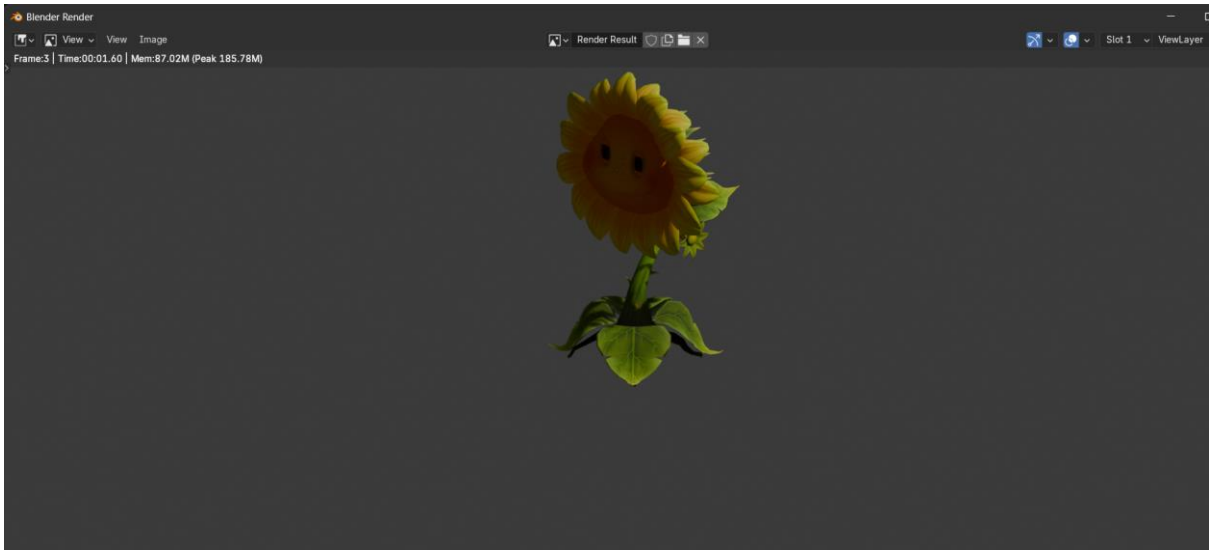
Step 8: Lighting and Camera

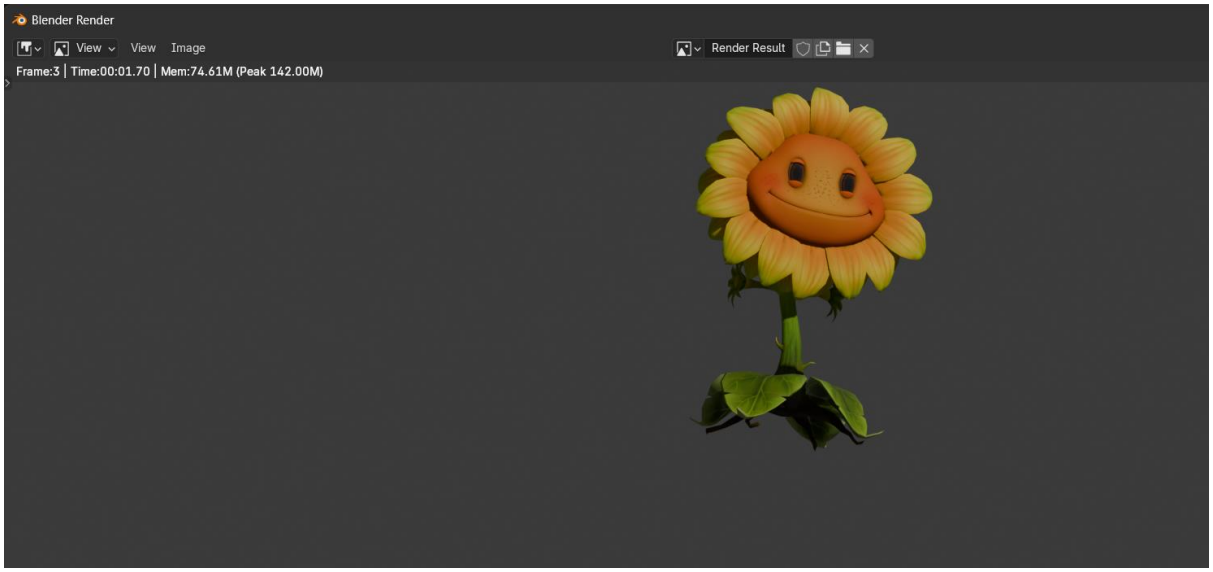
1. Shift + A → Light → Sun or Area
2. Position the light to shine on your flower.
3. Shift + A → Camera → Position to frame the flower.
4. Press 0 on numpad to view through camera.

Step 9: Render

1. Go to **Render Properties** (camera icon).
2. Choose **Eevee** or **Cycles** as render engine.
3. Press F12 to render the image.
4. Save the render (Image → Save As).

RENDERING RESULT:





RESULT:

Thus the above two object are 3d object we can rotate around 360 degree of any direction but I have given two rotation of the object as screenshot

MY CONTRIBUTION: (ANDROID APPLICATION) **- Arpit dixit (22MIC0028)**

The code below shows integration of plant Augmented Reality (AR) view based with IOT (internet of things) devices and code, which shows the plant with emotion:

- Happy plant AR when the reading range from IOT devices are good and plant is in good condition.
- Sad plant AR when plant condition is not good which we obtain from the IOT device.

The code below uses WebView using chrome web browser after integrating IOT code (Python Code) after taking various permission like camera, internet and file storage.

Instead of creating manual view we use web view here for IOT integration

Code Language: Java (Backend), XML (Frontend)

FRONTEND CODE: (MAIN_ACTIVITY.XML)

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
```

```
    <WebView
        android:id="@+id/webView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

BACKEND CODE: (MAINACTIVITY.JAVA)

```
package com.example.webview;
import android.Manifest;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.webkit.PermissionRequest;
import android.webkit.WebChromeClient;
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;
public class MainActivity extends AppCompatActivity {
    private static final int CAMERA_PERMISSION_REQUEST_CODE = 1;
    private WebView webView;
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        webView = findViewById(R.id.webView);
        webView.setWebViewClient(new WebViewClient());
        webView.setWebChromeClient(new WebChromeClient() {
            @Override
            public void onPermissionRequest(final PermissionRequest request) {
                runOnUiThread(() -> request.grant(request.getResources())); // Grant
camera permission to WebView
            }
        });
        WebSettings webSettings = webView.getSettings();
        webSettings.setJavaScriptEnabled(true);
        webSettings.setMediaPlaybackRequiresUserGesture(false);
        webSettings.setDomStorageEnabled(true);
        webSettings.setAllowFileAccess(true);
        webSettings.setAllowContentAccess(true);
        webSettings.setDatabaseEnabled(true);
        // Check and request camera permission
        if (ContextCompat.checkSelfPermission(this,
Manifest.permission.CAMERA)
                != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this,
                    new String[]{Manifest.permission.CAMERA},
CAMERA_PERMISSION_REQUEST_CODE);
        } else {
```

```
            loadWebView();
        }
    }
    private void loadWebView() {
        webView.loadUrl("https://vit.8thwall.app/plantar/");
    }
    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions,
                                           @NonNull int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions,
grantResults);
        if (requestCode == CAMERA_PERMISSION_REQUEST_CODE) {
            if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                loadWebView(); // Reload WebView if permission is granted
            }
        }
    }
}
```

OUTPUT: (AR AUGMENTATION):

MOBILE APP SCREENSHOT:


11:45
60%