

IoT-Based Smart Home Automation and Surveillance System

-VEDANT MAHADIK



A. Abstract

This project presents an IoT based smart home automation platform which integrates three inter related sub systems: Automated watering of plants, intelligence home environment and the highest level of surveillance are the key features of this system. Using combinations of different microcontrollers such as ESP8266, Arduino Uno and ESP32-CAM with other assorted sensors and actuators, the system provides a flexible and inexpensive design solution for the creation of a smart home. Technology used include Blynk Cloud, Alexa and a locally hosted Flask server for face detection algorithms. Media Pipe takes care of facial recognition while Cloudinary manages secure storage of cloud images.

Built to work wirelessly with little manual intervention, this system delivers value on plant health, home safety, and electric energy-providing wise use in home appliances. Relay operation is remote controlled through the internet using NodeMCU, acting on environmental factors. The Flask backend gives the

surveillance system AI, only images used to indicate the presence of a human being are saved for improved storage later. Each aspect of the system communicates effortlessly and is manageable through mobile devices or voice assistants. Unlike traditional commercial smart home solutions, this project is distinguished by its flexibility, scalability and affordability.

B. Keywords

- IoT
- Smart Home Automation
- ESP8266
- ESP32-CAM
- Blynk
- Alexa
- Sensors
- Remote Access
- Face Detection
- Clouinary
- Flask Server
- Voice Control

C. Introduction

Smart homes are contemporary homes in which appliances and systems are connected to the Internet for greater efficiency, security, and comfort. In a smart home, the sensors and actuators will be configured to form a network, and that network will have communication capabilities with one another and, possibly, also with the user's device (e.g. smartphone or voice assistant), creating an intelligent living environment.

Smart home systems are on the rise like never before due to improvements in IoT capabilities. Current systems like Google Nest, Ring, and Amazon Smart Home centralize control of lighting, heating, security cameras, etc. through operation with associated mobile apps and voice assistants. However, they have high installation and maintenance costs, are locked into a proprietary ecosystem of interchangeable systems, and thus lack customizability or extensibility. Several academic and DIY projects have explored low-cost smart

home solutions using microcontrollers like Arduino, ESP8266, and Raspberry Pi. These efforts typically focus on maintaining discrete capabilities on concepts related to home automation, plant irrigation, and security or surveillance. Many of these attempts result in single-purpose systems and therefore do not cover more than a single domain.

This project bridges that gap by integrating multiple smart systems—plant irrigation, environmental monitoring, and intelligent surveillance—into one unified framework. Unlike existing solutions that rely solely on cloud services or local processing, this project uses a hybrid model. This project proposes open-source platforms, low-cost hardware and expandable architecture and design as a low-cost, configurable, expandable and customizable smart home system.

D. Objectives

- Design and build an smart plant watering system using IoT with automated and manual control modes to water plants based on soil moisture.
- Monitor and send environmental parameters including temperature, humidity, flame, and gas to cloud server
- Enable device control through both mobile application (remote access) and voice commands
- Implement a smart surveillance system with real-time streaming and AI-based face detection with both local and cloud level storage
- Allow notifications and alerts for security and environmental effects

E. Theory/Methodology/Design

The complete system is divided into three interconnected subsystems

System 1: Smart Plant Watering System :-

The plant watering subsystem utilizes two capacitive soil moisture sensors (v2.0), which are inserted in the pots. Capacitive sensors can detect volumetric water content without the risk of corrosion. An Arduino Uno reads both analog values of moisture every second, then compares the moisture readings to pre-defined threshold values. Finally, the soil moisture value is sent to an ESP8266 via serial communication; the ESP pushes this value to the Blynk cloud. If the moisture falls below a threshold value of 45%, the relay is turned on by the Arduino to provide a 12V DC power supply to the water pump and 24V DC to

open a solenoid valve, watering will continue until the moisture level is above a threshold of 60%. The system will send a notification within Blynk when the pump turns on or off. In the Blynk app, toggle buttons allow the user to turn the motor and solenoid valve on or off remotely (manually). The system also monitors the level of water in the pump tank, using an HC-SR04 ultrasonic sensor. If it gets too low, the system will notify the user and also prevent the pump from dry running. This closed-loop control system allows the watering subsystem to monitor and optimally hydrate the plants without wasting unnecessary water.

System 2: Home Automation and Environment Monitoring :-

This subsystem is a monitoring system for home safety and comfort parameters. It uses an ESP8266 to read three sensors- a flame detector, an MQ-2 gas sensor and a DHT11 temperature/humidity sensor, commonly used in IoT fire/gas systems. The flame detector detects infrared from open flames, while the MQ-2 detects detectable flammable gases (LPG, methane, smoke). The ESP8266 reads the sensors and continually publishes their values to Blynk. There are two 5V relays connected to a 220V AC fan and AC light. In the Blynk app, toggle buttons are provided to remotely turn the fan and light on or off. The system can also use voice commands ("Alexa, turn on fan/light") via Sinric Pro. If either the flame or gas sensor exceeds a threshold of safety, the ESP8266 reads that an alert is required and, triggers a Blynk push notification ("Flame detected!" or "Gas leak!") to the user. The fan or light can also be activated automatically in response to detections, if desired.

System 3: Home Surveillance :-

The surveillance subsystem is using ESP32-CAM modules with two different camera sensor modules: OV3660 and OV2640 which give 3MP and 2MP, respectively. The Wi-Fi capabilities of the ESP32 have given flexibility in design and installation, as the subsystem can adapt to three different configurations.

- **Local Camera 1 (Local IP Streaming)** - One ESP32-CAM is using the Arduino "CameraWebServer" code, which makes an IP address on the home LAN accessible over the same Wi-Fi, so anyone can watch a live MJPEG stream whoever on the same Wi-Fi. This acts as a local security camera of sorts.

- **Remote Camera 1 (Flask + MediaPipe + SD card)** - A second ESP32-CAM captures a JPEG image every 5 seconds and sends it as an HTTP POST to a Flask backend server hosted on Render.com. This flask backend also interfaces Google MediaPipe Face Detection (an ultra fast, multi-face detection solution) as a python library to analyse each image sent from ESP32-CAM subsystems. If a face is detected then the Flask server sends the Json response back to camera module which saves the image on an SD card, and the server will update a web page showcasing each the capture in real time.
- **Remote Camera 2 (Flask + MediaPipe + Cloudinary):** A third ESP32-CAM posts images to a different Flask server in the same way the other ESP32 cameras integrate with cloud services. The difference is that if a face is detected, the server shares the image with Cloudinary (a cloud image service). This means all of the detected faces storage were centralised to one location. MediaPipe allows real-time detection on limited hardware and, with Cloudinary, we can store images off-device.

Voice Control Integration :-

Subsystems 1 and 2 implement Alexa voice control via Sinric Pro. An account with Sinric Pro is created and that account defines virtual devices that correspond with our water pump, valve, fan and light. Our ESP8266 firmware connects to this service via the Sinric Pro library. When the user says "Alexa, turn on the pump," Amazon sends the command to Sinric Pro, which sends the command back to our ESP8266 device. The code contains a callback function that turns on the relay that corresponds with the command received. Therefore, voice commands can control any device connected to a relay with no differences in the process. Sinric Pro acts as a cloud bridge between Alexa and the ESP8266.

F. Block Diagram

An overview of the architecture is shown below. Each subsystem (plant watering, home monitoring, surveillance) has sensors and actuators connected to microcontrollers (Arduino Uno or ESP8266/ESP32) and communicates over Wi-Fi to cloud services. The user interacts with the system via the Blynk mobile app (which is a custom dashboard with gauges, buttons, and notifications) and via voice commands through Alexa (using Sinric Pro).

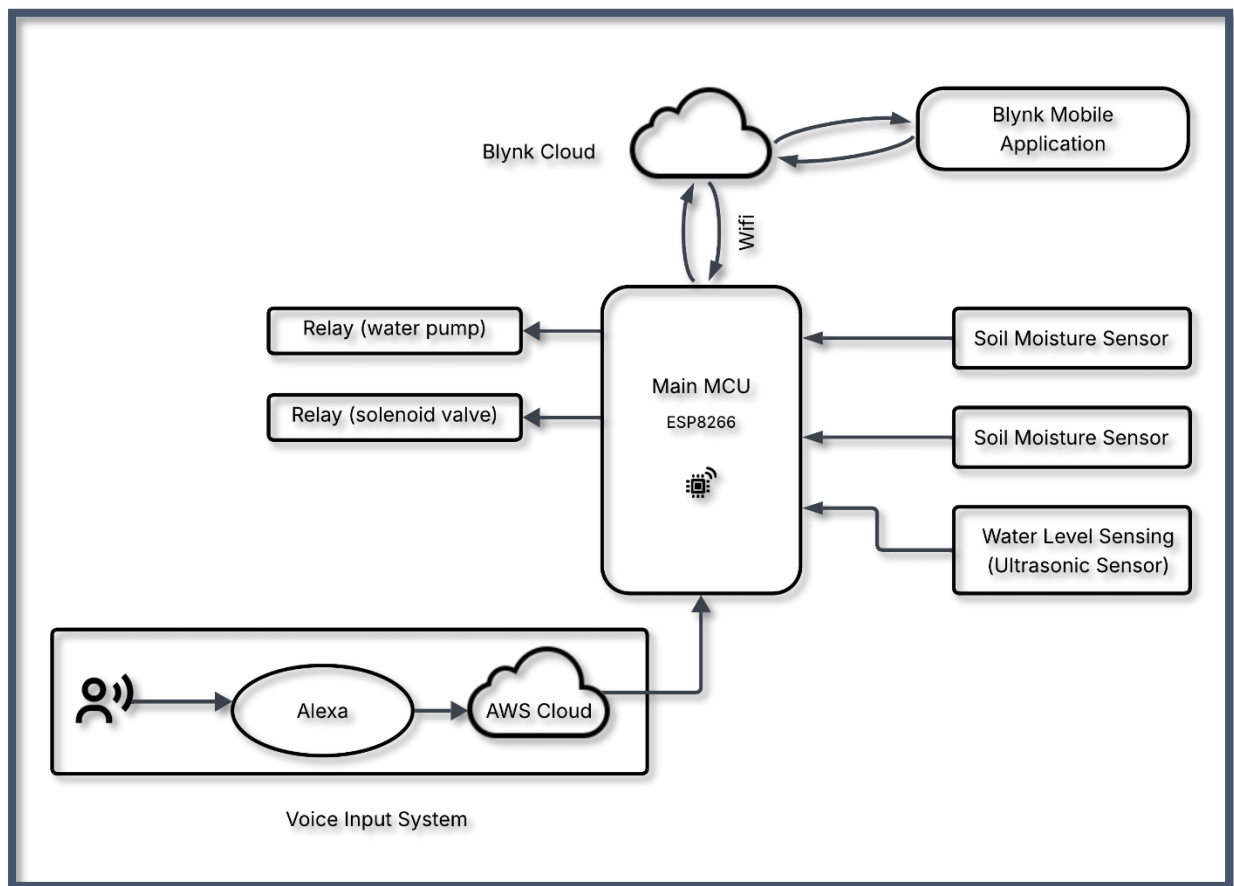


Fig 1 : System 1 - Smart Plant Watering System

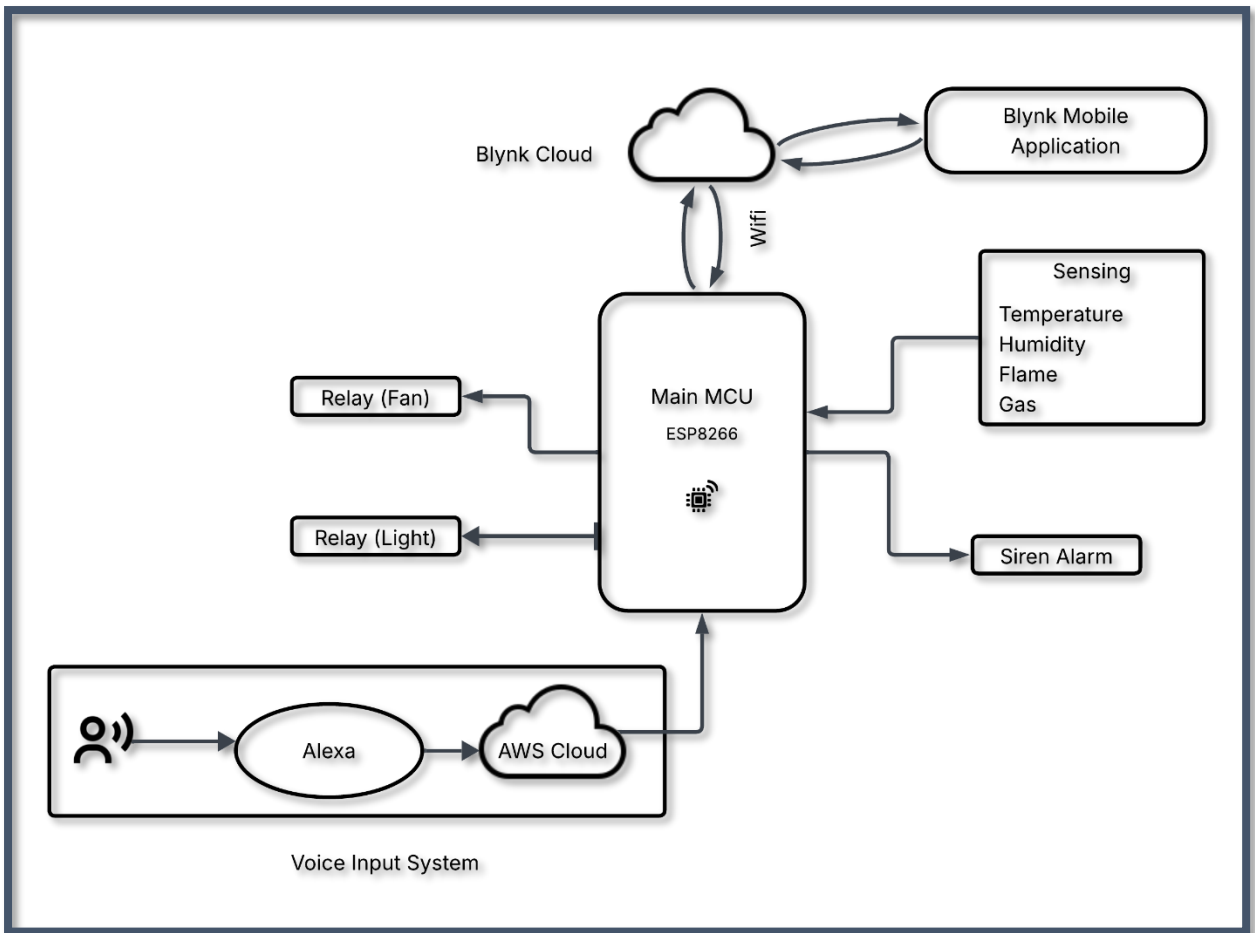


Fig 2 : System 2 - Home Automation and Environment Monitoring

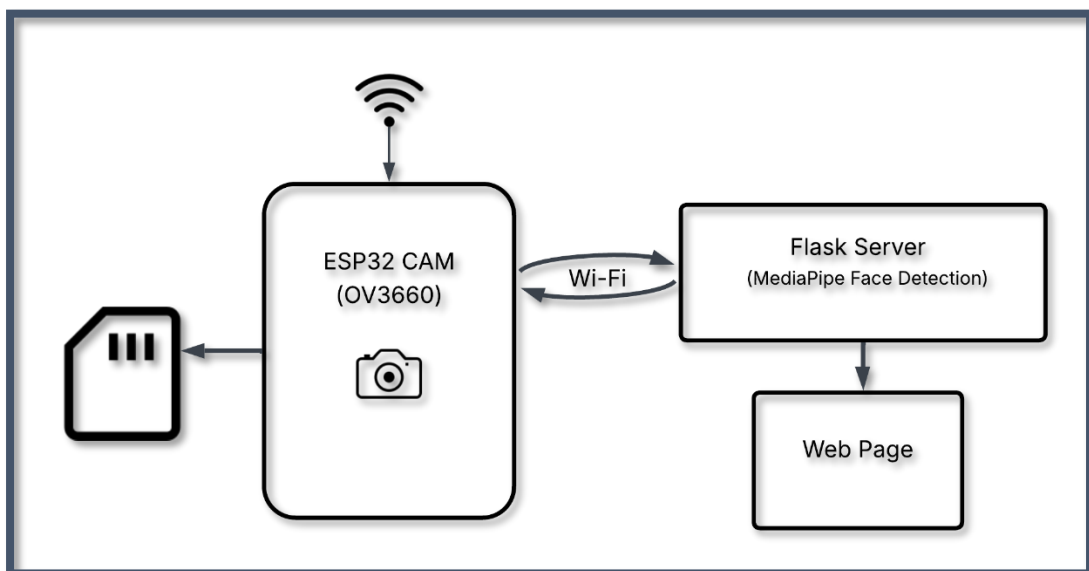


Fig 3 : System 3 - Remote Camera 1 (Flask + MediaPipe + SD card)

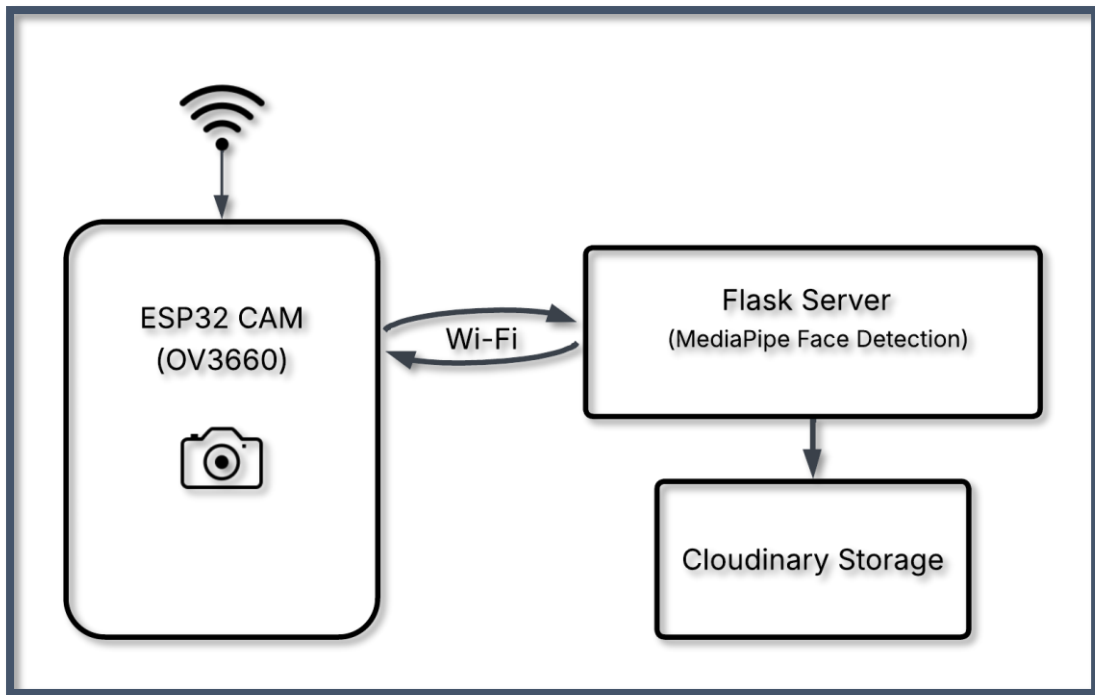


Fig 4 : System 3 - Remote Camera 2 (Flask + MediaPipe + Cloudinary)

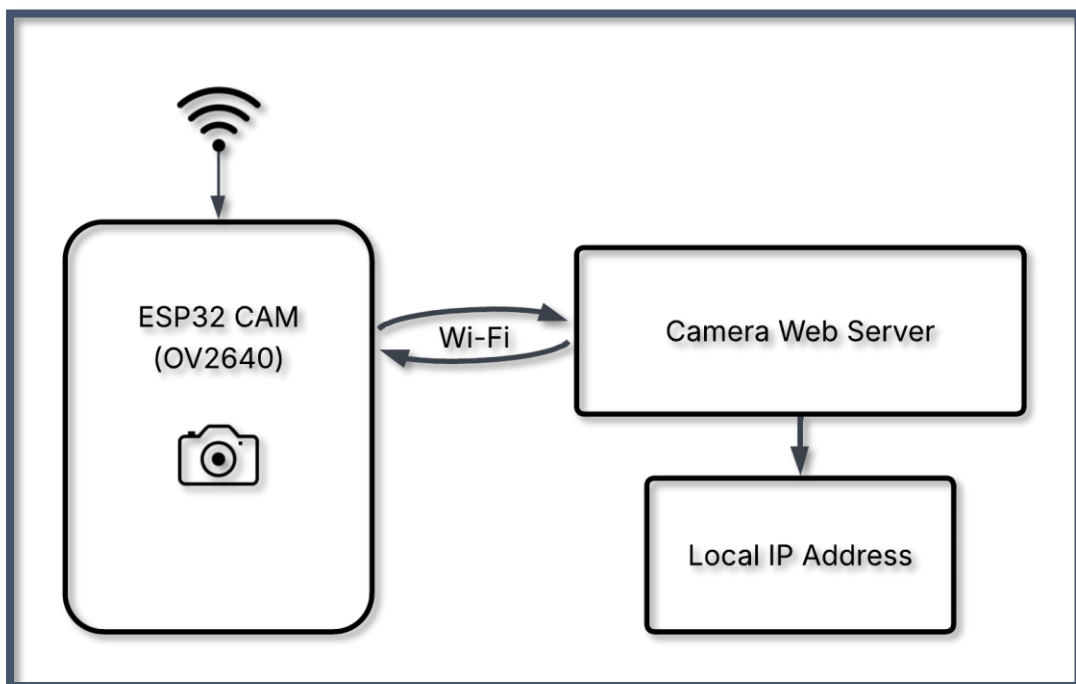


Fig 5 : System 3 - Local Camera 1 (Local IP Streaming)

G. Flowchart

H. Materials/Components Used

Component Name	Specification	Purpose
Arduino Uno	ATmega328P microcontroller	Reads soil moisture sensors
NodeMCU	ESP8266	Controls all the communication
Soil Moisture Sensor	Capacitive (V2.0)	Measures moisture content in plant soil (non-corrosive)
Ultrasonic Sensor	HC-SR04	Monitors water level in pump tank
Water Pump (12V DC)	Brushless Motor Submersible Solar Pump	Pumps water into plants when relay is activated
Solenoid Valve (24V DC)	NC (normally closed) valve	Controls water flow to plants (opens when powered)
Relay Modules	5V DC	Control switching states
Power Adapter	12V DC	Supply the power
5V DC Supply	LM7805	12V DC to 5V DC converter
Flame Sensor Module	Infrared photodiode array	Detects flames/fire (for safety alerts)
Gas Sensor	MQ-2	Senses combustible gases (LPG, methane, smoke)
Temperature/Humidity Sensor	DHT11	Measures temperature and humidity
Fan	220V AC Table Fan	Provides Ventilation and Cool air
Bulb	Household Bulb	Room light control
Camera Module	ESP32 cam / m5stack (OV3660/OV2640)	Captures images for surveillance, streams video (local IP camera)
Power Bank	5V DC output	Provide constant power supply

SD Card	16GB	Stores images on ESP32-CAM when faces are detected
Connecting Wires / Breadboard/ Brown Board	----	Prototyping and connections between components
Alexa	3 rd Generation	Voice Control

I. Code Developed/Used

System 1: Smart Plant Watering System :- (ESP8266 Code)

```
// Defining Blynk template and connected device
#define BLYNK_TEMPLATE_ID "TMPL3bAoM4LTQ"
#define BLYNK_TEMPLATE_NAME "Home Automation"
#define BLYNK_AUTH_TOKEN "m99R1no34dgl4rMozD1GBvo5iU7BT7Wy"

#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>
#include <SinricPro.h>
#include <SinricProSwitch.h>
#include <SoftwareSerial.h>

// WiFi credentials
char ssid[] = "MumbaiGalaxy";
char pass[] = "12345678";

// Moisture relay pins
#define RELAY3 14 // D5
#define RELAY4 12 // D6

// Ultrasonic sensor pins
#define TRIG_PIN 5 // D1
#define ECHO_PIN 4 // D2

// SinricPro credentials
#define APP_KEY "4b02bdf6-a926-405f-a4f7-b72cb52370a6"
#define APP_SECRET "dd2bdc67-ff66-41c2-a0ef-89cea0c899d9-8b8443c4-c678-439c-a8ab-35a2dd38c684"
#define DEVICE_ID_3 "67f2c6b2dc4a25d5c3a30d57" // Relay3

// Thresholds
#define MOISTURE_ON_THRESHOLD 45
#define MOISTURE_OFF_THRESHOLD 60
#define TANK_HEIGHT_CM 27
#define WATER_LEVEL_THRESHOLD 10

// Global variables
int m3 = 0, m4 = 0;
bool autoRelay3 = false, autoRelay4 = false;
bool waterNotified = false;
```

```

SoftwareSerial mySerial(13, 15); // RX = D7, TX = D8
String inputString = "";

void setupRelays() {
    pinMode(RELAY3, OUTPUT);
    pinMode(RELAY4, OUTPUT);
    digitalWrite(RELAY3, HIGH);
    digitalWrite(RELAY4, HIGH);
}

// Manual button control of relays by blynk app
BLYNK_WRITE(V0) { digitalWrite(RELAY3, param.asInt() ? LOW : HIGH); }
BLYNK_WRITE(V1) { digitalWrite(RELAY4, param.asInt() ? LOW : HIGH); }

// Voice control of relay 3
bool onPowerState3(const String &deviceId, bool state) {
    if (deviceId == DEVICE_ID_3) digitalWrite(RELAY3, state ? LOW : HIGH);
    return true;
}

void setupSinric() {
    SinricProSwitch &sw3 = SinricPro[DEVICE_ID_3];
    sw3.onPowerState(onPowerState3);
    SinricPro.begin(APP_KEY, APP_SECRET);
    SinricPro.restoreDeviceStates(true);
}

void setupUltrasonic() {
    pinMode(TRIG_PIN, OUTPUT);
    pinMode(ECHO_PIN, INPUT);
}

void setup() {
    Serial.begin(9600);
    mySerial.begin(9600);
    setupRelays();
    setupUltrasonic();
    Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
    setupSinric();
}

```

```

void loop() {
    Blynk.run();
    SinricPro.handle();
    readMoistureData();
    measureWaterLevel();
}

// Reads coming moisture data on serial from arduino UNO
void readMoistureData() {
    while (mySerial.available()) {
        char c = mySerial.read();
        if (c == '\n' || c == '\r') {
            if (inputString.length() > 0) {
                sscanf(inputString.c_str(), "%d,%d", &m3, &m4);
                inputString = "";
                Blynk.virtualWrite(V4, m3);
                Blynk.virtualWrite(V5, m4);
                autoControl();
            }
        } else {
            inputString += c;
        }
    }
}

// Control of relays according to set thresholds
void autoControl() {
    if (m3 < MOISTURE_ON_THRESHOLD && !autoRelay3) {
        digitalWrite(RELAY3, LOW); autoRelay3 = true;
    } else if (m3 > MOISTURE_OFF_THRESHOLD && autoRelay3) {
        digitalWrite(RELAY3, HIGH); autoRelay3 = false;
    }

    if (m4 < MOISTURE_ON_THRESHOLD && !autoRelay4) {
        digitalWrite(RELAY4, LOW); autoRelay4 = true;
    } else if (m4 > MOISTURE_OFF_THRESHOLD && autoRelay4) {
        digitalWrite(RELAY4, HIGH); autoRelay4 = false;
    }
}

```

```
// Measures water level in pump tank
void measureWaterLevel() {
    digitalWrite(TRIG_PIN, LOW);
    delayMicroseconds(2);
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);

    long duration = pulseIn(ECHO_PIN, HIGH);
    int distance = duration * 0.0344 / 2;
    int water_level = TANK_HEIGHT_CM - distance;

    if (water_level < 0) water_level = 0;
    if (water_level > TANK_HEIGHT_CM) water_level = TANK_HEIGHT_CM;

    Blynk.virtualWrite(V10, water_level);

    if (water_level < WATER_LEVEL_THRESHOLD && !waterNotified) {
        Blynk.logEvent("low_water", "Water level is low, fill it quickly!!");
        waterNotified = true;
    } else if (water_level >= WATER_LEVEL_THRESHOLD && waterNotified) {
        waterNotified = false;
    }
}
```

System 1: Smart Plant Watering System :- (Arduino UNO Code)

```
#include <SoftwareSerial.h>

// Serial for making connection with ESP8266
SoftwareSerial espSerial(10, 11); // RX, TX (Uno's 10 = RX from ESP, 11 = TX to ESP)

void setup() {
  Serial.begin(9600);          // For debugging on Serial Monitor
  espSerial.begin(9600);       // Communication with ESP8266
}

void loop() {
  // Read and map moisture values
  int m3 = map(analogRead(A0), 1023, 0, 0, 100); // Moisture sensor 1
  int m4 = map(analogRead(A1), 1023, 0, 0, 100); // Moisture sensor 2

  // Send data to ESP in CSV format: m3,m4
  espSerial.print(m3);
  espSerial.print(",");
  espSerial.println(m4); // Send new line for each data packet

  //Also print to Serial Monitor for debugging
  Serial.print("M3: "); Serial.print(m3); Serial.print("%, ");
  Serial.print("M4: "); Serial.print(m4); Serial.print("%, ");

  delay(2000); // Send data every 2 seconds
}
```

System 2: Home Automation and Environment Monitoring

```
// Defining Blynk template and connected device
#define BLYNK_TEMPLATE_ID "TMPL3bAoM4LTQ"
#define BLYNK_TEMPLATE_NAME "Home Automation"
#define BLYNK_AUTH_TOKEN "m99R1no34dgl4rMozD1GBvo5iU7BT7Wy"

#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>
#include <DHT.h>
#include <SinricPro.h>
#include <SinricProSwitch.h>

// DHT sensor
#define DHTPIN 5
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

// Gas sensor pin
const int gasPin = A0;

// Relay pins
#define relay1Pin 13
#define relay2Pin 15

// Flame sensor pin
#define flamePin 4

// SinricPro credentials
#define DEVICE_ID_1 "67f2c5fd8ed485694c0ba0b1" // relay 1
#define DEVICE_ID_2 "67f2c683bddfc53e33cae0f8" // relay 2
#define APP_KEY     "4b02bdf6-a926-405f-a4f7-b72cb52370a6"
#define APP_SECRET  "dd2bdc67-ff66-41c2-a0ef-89cea0c899d9-8b8443c4-c678-439c-a8ab-35a2dd38c684"

// WiFi credentials
char ssid[] = "MumbaiGalaxy";
char pass[] = "12345678";

BlynkTimer timer;
```

```

// Voice control of relays 1 and 2
bool onPowerState(const String &deviceId, bool state) {
    if (deviceId == DEVICE_ID_1) digitalWrite(relay1Pin, state ? LOW : HIGH);
    else if (deviceId == DEVICE_ID_2) digitalWrite(relay2Pin, state ? LOW : HIGH);
    return true;
}

void setupSinric() {
    SinricProSwitch &sw1 = SinricPro[DEVICE_ID_1];
    SinricProSwitch &sw2 = SinricPro[DEVICE_ID_2];
    sw1.onPowerState(onPowerState);
    sw2.onPowerState(onPowerState);
    SinricPro.begin(APP_KEY, APP_SECRET);
    SinricPro.restoreDeviceStates(true);
}

void setup() {
    Serial.begin(115200);
    Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
    pinMode(relay1Pin, OUTPUT);
    pinMode(relay2Pin, OUTPUT);
    pinMode(flamePin, INPUT);
    digitalWrite(relay1Pin, LOW);
    digitalWrite(relay2Pin, LOW);
    dht.begin();
    setupSinric();
    timer.setInterval(1000L, sendDataToBlynk);
}

void loop() {
    Blynk.run();
    timer.run();
    SinricPro.handle();
}

```



```

// Sending sensors data to blynk cloud
void sendSensorData() {
    float h = dht.readHumidity();
    float t = dht.readTemperature();
    int gasLevel = analogRead(gasPin);
    int flameStatus = digitalRead(flamePin); // 1 = Flame Detected, 0 = No Flame

    Serial.print("Humidity: "); Serial.print(h);
    Serial.print(" %\tTemperature: "); Serial.print(t);
    Serial.print(" °C\tGas Level: "); Serial.print(gasLevel);
    Serial.print("\tFlame Status: "); Serial.println(flameStatus == 0 ? "No Flame" : "Flame Detected!");

    // Checks for empty temperature and humidity sensor data
    if (!isnan(h) && !isnan(t)) {
        Blynk.virtualWrite(V2, h);
        Blynk.virtualWrite(V3, t);
    } else {
        Serial.println("Failed to read from DHT11 sensor!");
    }

    Blynk.virtualWrite(V7, gasLevel);
    Blynk.virtualWrite(V6, flameStatus);
}

// Manual button control of relay 1 by blynk app
BLYNK_WRITE(V8) {
    int relay1State = param.asInt();
    digitalWrite(relay1Pin, relay1State == 0 ? HIGH : LOW);
}

// Manual button control of relay 2 by blynk app
BLYNK_WRITE(V9) {
    int relay2State = param.asInt();
    digitalWrite(relay2Pin, relay2State == 0 ? HIGH : LOW);
}

// Uploading data to blynk cloud
void sendDataToBlynk() {
    sendSensorData();
}

```

System 3: Home Surveillance :- (Local Camera 1)

```

#include "esp_camera.h"
#include <WiFi.h>

// Set camera model
#define CAMERA_MODEL_AI_THINKER

// Importing camera pins
#include "camera_pins.h"

// Enter your WiFi credentials
const char *ssid = "Cepheus2023";
const char *password = "11223344";

void startCameraServer();
void setupLedFlash(int pin);

void setup() {
    Serial.begin(115200);
    Serial.setDebugOutput(true);
    Serial.println();

    camera_config_t config;
    config.ledc_channel = LEDC_CHANNEL_0;
    config.ledc_timer = LEDC_TIMER_0;
    config.pin_d0 = Y2_GPIO_NUM;
    config.pin_d1 = Y3_GPIO_NUM;
    config.pin_d2 = Y4_GPIO_NUM;
    config.pin_d3 = Y5_GPIO_NUM;
    config.pin_d4 = Y6_GPIO_NUM;
    config.pin_d5 = Y7_GPIO_NUM;
    config.pin_d6 = Y8_GPIO_NUM;
    config.pin_d7 = Y9_GPIO_NUM;
    config.pin_xclk = XCLK_GPIO_NUM;
    config.pin_pclk = PCLK_GPIO_NUM;
    config.pin_vsync = VSYNC_GPIO_NUM;
    config.pin_href = HREF_GPIO_NUM;
    config.pin_sccb_sda = SIOD_GPIO_NUM;

```

```

config.pin_sccb_scl = SIOC_GPIO_NUM;
config.pin_pwdn = PWDN_GPIO_NUM;
config.pin_reset = RESET_GPIO_NUM;
config.xclk_freq_hz = 20000000;
config.frame_size = FRAMESIZE_QVGA;
config.pixel_format = PIXFORMAT_RGB565;
config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
config.fb_location = CAMERA_FB_IN_PSRAM;
config.jpeg_quality = 12;
config.fb_count = 1;

// if PSRAM IC present, init with UXGA resolution and higher JPEG quality
//                               for larger pre-allocated frame buffer.
if (config.pixel_format == PIXFORMAT_JPEG) {
    if (psramFound()) {
        config.jpeg_quality = 10;
        config.fb_count = 2;
        config.grab_mode = CAMERA_GRAB_LATEST;
    } else {
        // Limit the frame size when PSRAM is not available
        config.frame_size = FRAMESIZE_SVGA;
        config.fb_location = CAMERA_FB_IN_DRAM;
    }
} else {
    // Best option for face detection/recognition
    config.frame_size = FRAMESIZE_240X240;
}
#if CONFIG_IDF_TARGET_ESP32S3
    config.fb_count = 2;
#endif

}

#if defined(CAMERA_MODEL_ESP_EYE)
    pinMode(13, INPUT_PULLUP);
    pinMode(14, INPUT_PULLUP);
#endif

```

```

// camera init
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
}

sensor_t *s = esp_camera_sensor_get();
// initial sensors are flipped vertically and colors are a bit saturated
if (s->id.PID == OV3660_PID) {
    s->set_vflip(s, 1);          // flip it back
    s->set_brightness(s, 1);    // up the brightness just a bit
    s->set_saturation(s, -2);    // lower the saturation
}
// drop down frame size for higher initial frame rate
if (config.pixel_format == PIXFORMAT_JPEG) {
    s->set_framesize(s, FRAMESIZE_QVGA);
}

#ifdef CAMERA_MODEL_M5STACK_WIDE || defined(CAMERA_MODEL_M5STACK_ESP32CAM)
    s->set_vflip(s, 1);
    s->set_hmirror(s, 1);
#endif

#ifdef CAMERA_MODEL_ESP32S3_EYE
    s->set_vflip(s, 1);
#endif

// Setup LED Flash if LED pin is defined in camera_pins.h
#ifdef LED_GPIO_NUM
    setupLedFlash(LED_GPIO_NUM);
#endif

WiFi.begin(ssid, password);
WiFi.setSleep(false);

```

```
Serial.print("WiFi connecting");
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");

startCameraServer();

Serial.print("Camera Ready! Use 'http://");
Serial.print(WiFi.localIP());
Serial.println("' to connect");
}

void loop() {
    delay(5000);
}
```

System 3: Home Surveillance :- (Remote Camera 1)

```
#include "esp_camera.h"
#include <WiFi.h>
#include <HTTPClient.h>
#include "FS.h"
#include "SD_MMC.h"

// Defining the esp32 cam module
#define CAMERA_MODEL_AI_THINKER

// Camera sensor pins
#define PWDN_GPIO_NUM    32
#define RESET_GPIO_NUM  -1
#define XCLK_GPIO_NUM    0
#define SIOD_GPIO_NUM    26
#define SIOC_GPIO_NUM    27

#define Y9_GPIO_NUM      35
#define Y8_GPIO_NUM      34
#define Y7_GPIO_NUM      39
#define Y6_GPIO_NUM      36
#define Y5_GPIO_NUM      21
#define Y4_GPIO_NUM      19
#define Y3_GPIO_NUM      18
#define Y2_GPIO_NUM       5
#define VSYNC_GPIO_NUM   25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22

// 4 for flash led or 33 for normal led
#define LED_GPIO_NUM      4

// WiFi credentials
const char* ssid = "Cepheus2023";
const char* password = "11223344";

// Flask server URL
const char* serverUrl = "https://home-esp.onrender.com/upload";
```

```
void setup() {  
    Serial.begin(115200);  
    delay(1000);  
  
    // Connect to WiFi  
    WiFi.begin(ssid, password);  
    Serial.print("Connecting to WiFi");  
    while (WiFi.status() != WL_CONNECTED) {  
        delay(500);  
        Serial.print(".");  
    }  
    Serial.println("\nWiFi connected");  
  
    // Camera configuration  
    camera_config_t config;  
    config.ledc_channel = LEDC_CHANNEL_0;  
    config.ledc_timer = LEDC_TIMER_0;  
    config.pin_d0 = Y2_GPIO_NUM;  
    config.pin_d1 = Y3_GPIO_NUM;  
    config.pin_d2 = Y4_GPIO_NUM;  
    config.pin_d3 = Y5_GPIO_NUM;  
    config.pin_d4 = Y6_GPIO_NUM;  
    config.pin_d5 = Y7_GPIO_NUM;  
    config.pin_d6 = Y8_GPIO_NUM;  
    config.pin_d7 = Y9_GPIO_NUM;  
    config.pin_xclk = XCLK_GPIO_NUM;  
    config.pin_pclk = PCLK_GPIO_NUM;  
    config.pin_vsync = VSYNC_GPIO_NUM;  
    config.pin_href = HREF_GPIO_NUM;  
    config.pin_sscb_sda = SIOD_GPIO_NUM;  
    config.pin_sscb_scl = SIOC_GPIO_NUM;  
    config.pin_pwdn = PWDN_GPIO_NUM;  
    config.pin_reset = RESET_GPIO_NUM;  
    config.xclk_freq_hz = 20000000;  
    config.pixel_format = PIXFORMAT_JPEG;
```

```

if(psramFound()){
    config.frame_size = FRAMESIZE_XGA; // Higher resolution
    config.jpeg_quality = 8;           // Better image quality
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_VGA;
    config.jpeg_quality = 8;
    config.fb_count = 1;
}

// Initialize camera
if (esp_camera_init(&config) != ESP_OK) {
    Serial.println("Camera init failed");
    return;
}

// Apply color corrections
sensor_t * s = esp_camera_sensor_get();
s->set_brightness(s, 2); // -2 to 2
s->set_contrast(s, 1); // -2 to 2
s->set_saturation(s, 2); // -2 to 2
s->set_whitebal(s, 1); // Enable auto white balance
s->set_awb_gain(s, 1); // Enable white balance gain
s->set_wb_mode(s, 0); // 0 = Auto white balance

// Initialize SD card
if (!SD_MMC.begin()) {
    Serial.println("SD Card Mount Failed");
    return;
}
if (SD_MMC.cardType() == CARD_NONE) {
    Serial.println("No SD card attached");
    return;
}
Serial.println("SD card initialized");

```



```

void loop() {
    // Check WiFi connection
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("WiFi disconnected, reconnecting...");
        WiFi.reconnect();
        delay(2000);
        return;
    }

    // Capture image
    camera_fb_t* fb = esp_camera_fb_get();
    if (!fb) {
        Serial.println("Camera capture failed");
        return;
    }

    // Send image to Flask server
    HTTPClient http;
    http.begin(serverUrl);
    http.addHeader("Content-Type", "image/jpeg");
    http.setTimeout(5000);

    int httpResponseCode = http.POST(fb->buf, fb->len);

    if (httpResponseCode == 200) {
        String response = http.getString();
        Serial.println("Server response: " + response);

        // Check if face is detected
        if (response.indexOf("\"face_detected\":true") >= 0) {
            String path = "/face_" + String(millis()) + ".jpg";
            File file = SD_MMC.open(path, FILE_WRITE);
            if (!file) {
                Serial.println("Failed to open file for writing");
            } else {
                file.write(fb->buf, fb->len);
                file.close();
                Serial.println("Saved image to: " + path);
            }
        } else {

```

```
        Serial.println("No face detected");
    }
} else {
    Serial.printf("HTTP POST failed with code: %d\n", httpResponseCode);
}

http.end();
esp_camera_fb_return(fb);

delay(5000); // Wait before next capture
}
```

System 3: Home Surveillance :- (Remote Camera 2)

```

#include "esp_camera.h"
#include <WiFi.h>
#include <HTTPClient.h>

// WiFi Credentials
const char* ssid = "Cepheus2023";
const char* password = "11223344";

// Flask Server URL
const char* serverUrl = "https://face-detection-flask-zkqh.onrender.com/upload";

// M5Stack Timer Camera F Pin Configuration (OV3660)
#define PWDN_GPIO_NUM    -1
#define RESET_GPIO_NUM   15
#define XCLK_GPIO_NUM     27
#define SIOD_GPIO_NUM     25
#define SIOC_GPIO_NUM     23

#define Y9_GPIO_NUM       19
#define Y8_GPIO_NUM       36
#define Y7_GPIO_NUM       18
#define Y6_GPIO_NUM       39
#define Y5_GPIO_NUM        5
#define Y4_GPIO_NUM       34
#define Y3_GPIO_NUM       35
#define Y2_GPIO_NUM       32
#define VSYNC_GPIO_NUM    22
#define HREF_GPIO_NUM     26
#define PCLK_GPIO_NUM     21

void setupCamera() {
    camera_config_t config;
    config.ledc_channel = LEDC_CHANNEL_0;
    config.ledc_timer = LEDC_TIMER_0;
    config.pin_d0 = Y2_GPIO_NUM;
    config.pin_d1 = Y3_GPIO_NUM;
    config.pin_d2 = Y4_GPIO_NUM;

```

```

config.pin_d3 = Y5_GPIO_NUM;
config.pin_d4 = Y6_GPIO_NUM;
config.pin_d5 = Y7_GPIO_NUM;
config.pin_d6 = Y8_GPIO_NUM;
config.pin_d7 = Y9_GPIO_NUM;
config.pin_xclk = XCLK_GPIO_NUM;
config.pin_pclk = PCLK_GPIO_NUM;
config.pin_vsync = VSYNC_GPIO_NUM;
config.pin_href = HREF_GPIO_NUM;
config.pin_sscb_sda = SIOD_GPIO_NUM;
config.pin_sscb_scl = SIOC_GPIO_NUM;
config.pin_pwdn = PWDN_GPIO_NUM;
config.pin_reset = RESET_GPIO_NUM;
config.xclk_freq_hz = 20000000;
config.pixel_format = PIXFORMAT_JPEG;

// Enhance image resolution and quality
config.frame_size = FRAMESIZE_VGA; // 640x480 resolution (better than QVGA)
config.jpeg_quality = 5;           // Better image quality (lower number is better)
config.fb_count = 2;               // 2 frame buffers for better performance

esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    Serial.printf("Camera init failed! Error: 0x%x", err);
    return;
}

// Get camera sensor settings
sensor_t * s = esp_camera_sensor_get();

// Apply image corrections (for OV3660 sensor)
s->set_brightness(s, 1); // -2 to 2
s->set_contrast(s, 1);   // -2 to 2
s->set_saturation(s, 2); // -2 to 2
s->set_whitebal(s, 1);   // Enable Auto White Balance
s->set_awb_gain(s, 1);   // Enable AWB Gain
s->set_wb_mode(s, 0);    // Auto white balance mode
s->set_sharpness(s, 2);  // Increase sharpness (if supported)

```

```

s->set_gain_ctrl(s, 1);      // Enable automatic gain
s->set_exposure_ctrl(s, 1); // Enable automatic exposure
s->set_gainceiling(s, (gainceiling_t)6); // Set higher gain ceiling for better low-light
}

void setup() {
  Serial.begin(115200);

  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500); Serial.print(".");
  }
  Serial.println("\nWiFi connected");

  setupCamera();
}

void loop() {
  camera_fb_t* fb = esp_camera_fb_get();
  if (!fb) {
    Serial.println("Camera capture failed");
    return;
  }

  HTTPClient http;
  http.begin(serverUrl);
  http.addHeader("Content-Type", "image/jpeg");

  int httpResponseCode = http.POST(fb->buf, fb->len);
  Serial.printf("HTTP POST Status: %d\n", httpResponseCode);

  http.end();
  esp_camera_fb_return(fb);

  delay(5000); // Capture every 10 seconds
}

```

J. Results and Discussion

The integrated IoT system is effective in its design goals. The capacitive sensors in the plant watering subsystem provided consistent moisture readings over time, with no degradation due to corrosion, the pump and valve turned on and off at the 45%/60% moisture threshold, keeping soil moisture contained. The data from the sensors shows up immediately in the Blynk app and manual watering from the app worked as intended. The low-water condition in the tank led to an immediate push notification and the pump turned off, all as it should.

In the home safety subsystem, the flame sensor and MQ-2 sensor reliably identified a small flame and low levels of test gas. The ESP8266 sent instantaneous notifications to the phone ("Flame detected!" or "Gas leak!"). This

indicates that real-time hazards can be alerted to through notifications. The DHT11 ambient readings appeared reasonable, and the fan/light relays behaved appropriately to the app and voice commands. Findings were aligned with peer-reviewed published IoT systems that use a flame and similar sensors in continuous fire/gas monitoring.

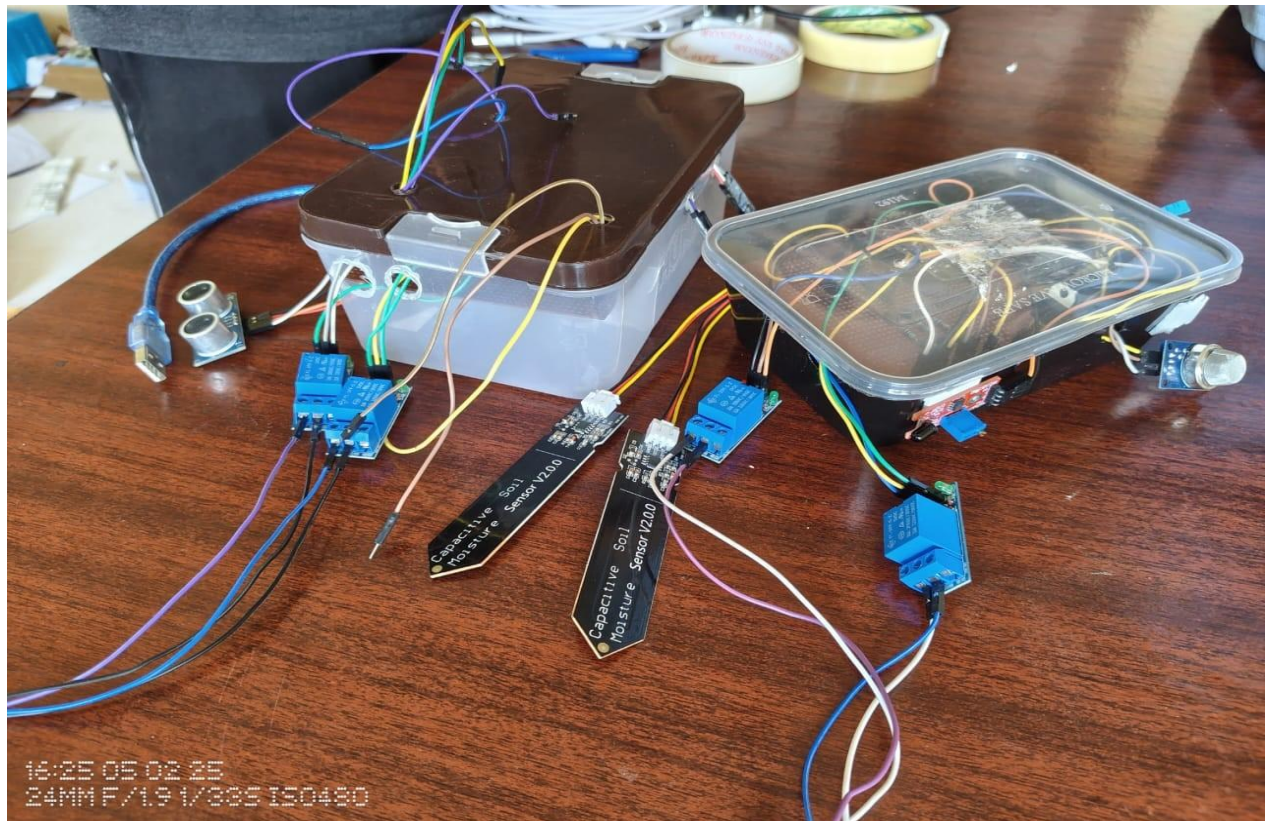
The surveillance sub-system worked as intended. The local ESP32-CAM provided a seamless live video stream via the browser, while the remote cameras were able to upload images to the Flask servers every five seconds. The MediaPipe face detector was able to identify a human face in the test images reliably (with sub-100mS detection time), and either saved the image on SD or uploaded it to Cloudinary as intended, for users to view the images they had captured on the web interface. No false detection was recorded during testing. This result aligns with the literature regarding smart surveillance, which identify that real-time detection and notifying the user is the focus.

Overall, the system was robust in testing. Network delays were negligible for local operations, though occasional 1–2 second lag occurred for remote image uploads. The Blynk dashboard remained synchronized with device states. Voice commands via Sinric Pro worked reliably (subject to Amazon Alexa's connectivity). Any limitations (such as Sinric Pro's device limit or Wi-Fi range) can be addressed in future work. These results validate that the combined IoT system – using sensors, relays, cloud apps, and voice AI – effectively automates watering, home safety, and security as intended. These results are consistent with published IoT hazard detection and surveillance solutions

K. Summary

The project shows an integrated approach to smart home automation with environmental monitoring, actions on devices, and surveillance. It is based around open-source hardware and software that is affordable, scalable and user-friendly. The successful remote access component, together with voice control, emphasizes the convenience for the user, and the modular position from several components integrates them into an expandable smart home.

L. User Guide



1. Take a 12V DC adapter and connect its 12V wire (mostly red) to the 12V pin given on the box, and connect the ground wire (mostly black) to the ground pin on the box. This will act as the power supply for the device to work. This step will be common for both devices (for the outdoor automatic watering system device 1 and the internal home automation system device 2).
2. For device 1(watering system), take the two moisture sensors provided and place them in the soil. Ensure that the sensor's circuit part is not in the soil, only the part below. Connect the two relays provided (relay 3 and 4) with the watering components like a motor (put it in some water storage) or a valve (attach it to the water tap and keep the tap open). Ensure that you provide proper power sources to the components as they need (12V or 24V). Take your Alexa and add the two components to the Alexa app, and provide the device IDs of the two relays to the code (it's already added for one of the relays, relay 3).
3. For device 2 (indoor home automation), connect the two relays provided (relays 1 and 2) with the home appliances you want to automate. Also, do the same process as done for Alexa in device 1(device IDs for relays 1 and 2 are provided in the code).
4. Blynk setup:
 - Sign in to the Blynk if you already have an account, or sign up to create a new account.

- Go to the developer zone and add a new template with a name you want, and then add a new device to it as per your name.
- In that device, you will be able to see your Blynk template ID and name, and add these to the code provided. Go back, where you can see your devices, from there you will get Blynk authentication token, so add that also to the code.

Firmware configuration



Template ID and Template Name should be declared at the very top of the firmware code.

```
#define BLYNK_TEMPLATE_ID "TMPL3cG2rQqt6"
#define BLYNK_TEMPLATE_NAME "FINALRELAY"
```

Devices

+ New Device

Start typing

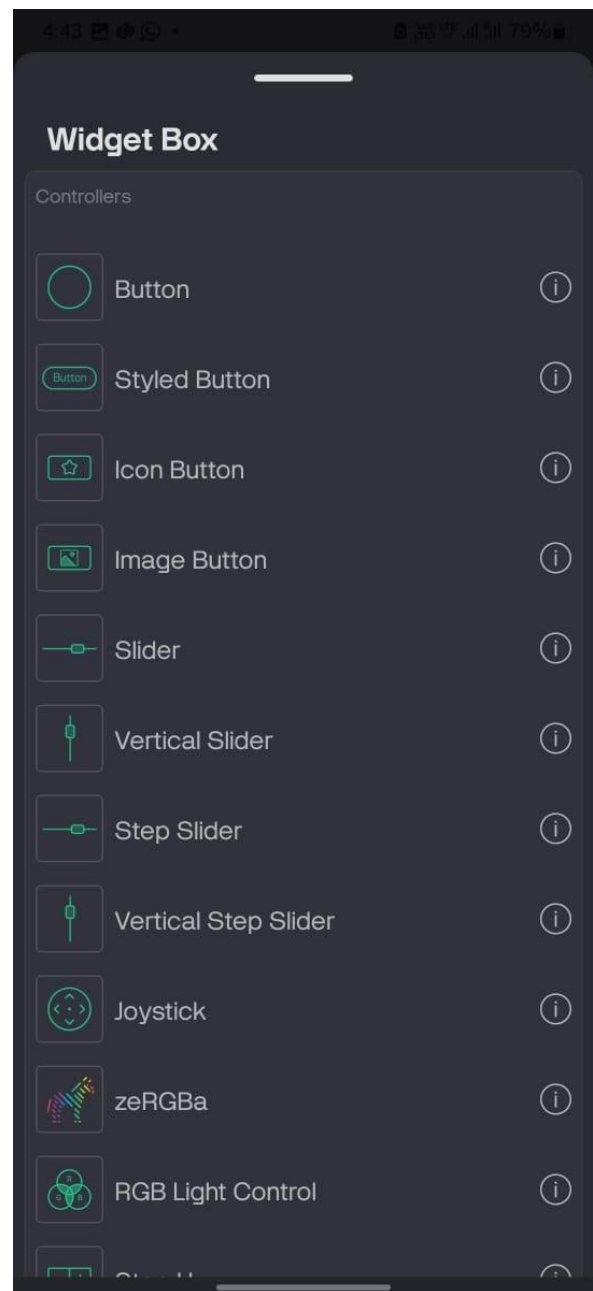
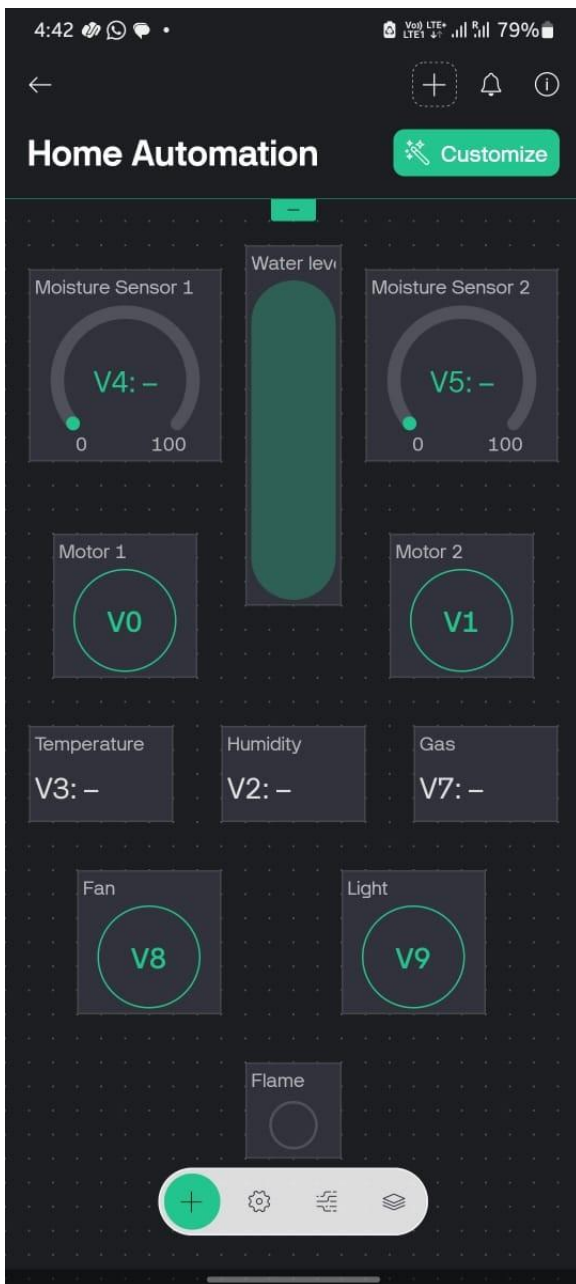
Add Filter

All 2 My devices

<input type="checkbox"/>	Name	Auth Token	Device Owner	Status	Last R	Actions
	CamSajal	e062b6w6EAedyFuxdVj3nVv37TBa...	mahadik.ravindra.22042@iitgoa.ac...	Offline		
	NEWDEVICE1	7jRq_uP8-krdLDDCio9un70j-cV67q...	mahadik.ravindra.22042@iitgoa.ac...	Inactive	3:45	

- In your device, go to the datastreams tab and then click on add datastream. Each time you add a new datastream, you will be asked to put the component name, set the virtual pin as provided in the code, enter the range of values (min., max., default), and then the unit. Don't forget to save it before exiting this tab.
- Take your mobile, open Blynk on it, sign in to your account, and then select your device in it. After signing in, you will be able to see all your devices and the datastreams you created earlier.
- In your device, click on the spanner symbol on the top right side, and there you will be able to add (click on +) various icons and scales to use them as a button, measuring bar, LED, etc. When you add an icon, you will be asked to assign a datastream to that icon from the list of datastreams that you created earlier. Ensure that you assign the datastreams correctly for proper use. The datastreams you select will be shown on the icon as shown

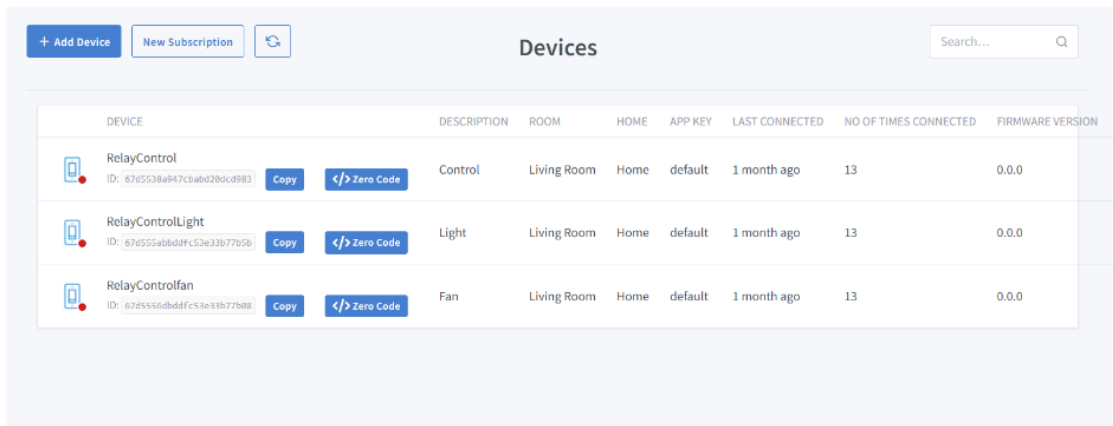
below. Do it for all the relays and the sensors you are using, and then exit the tab.



5. Alexa setup:

- Visit the Sinric Pro website and sign up. There, go to devices and new devices for as many devices as you want to control via voice command. Give a proper name to those devices.
- In your Alexa app, go to the skills and games tab, search for Sinric Pro, and then click on enable skill. Now, you will be asked to put Sinric Pro credentials in the tab, and after that, it will get connected to the Sinric Pro website.

- For the devices you add, you will get separate device IDs for each device, which you need to put in the code provided. Go to the credentials tab, and there you will get your App key and App Secret. Put these also in the code.



DEVICE	DESCRIPTION	ROOM	HOME	APP KEY	LAST CONNECTED	NO OF TIMES CONNECTED	FIRMWARE VERSION
RelayControl ID: 67d553ba947c3ab0280c0893	Control	Living Room	Home	default	1 month ago	13	0.0.0
RelayControlLight ID: 67d555abdbdfc53e33b77b5b	Light	Living Room	Home	default	1 month ago	13	0.0.0
RelayControlfan ID: 67d555abdbdfc53e33b77b5b	Fan	Living Room	Home	default	1 month ago	13	0.0.0

- Finally, when all the setup is done and you turn on your devices, it should show online on the Blynk and Alexa app. Once it is shown online, you are ready to automate your home!

1. Camera Access:

The cameras run automatically on power up so just give stable 5V DC power supply and wi-fi to cameras.

- To view the local IP camera stream, open a web browser on a device connected to the same Wi-Fi and navigate to the ESP32-CAM's IP address like <http://localaddress/>
- For remote camera 1, the Flask server web page will refresh with the latest detected-face images. Check the microSD cards in the ESP32-CAMs to see all saved captures. To see the latest capture, redirect to <https://home-esp.onrender.com/view>
- For remote camera 2, Check the Cloudinary console to see all saved captures when the face is detected.

2. General Notes:

Maintain stable Wi-Fi on all devices. If power cycled, all devices will connect with Blynk and Sinric after power is restored. All alerts (low water, flame and gas) are pushed to you immediately through the push notifications in the Blynk app. If setting modifications are needed (thresholds, Wi-Fi SSID, auth tokens), you may change the source code, run the code, and upload the code onto your devices. Be always cautious in your testing of gas or fires (only run tests in a safe and controlled manner and in proper benches).

M. Future Works

- **Scalable Protocols** - Instead of using the Blynk communication, Integrate MQTT or WebSocket-based messaging to improve scalability and reduce latency. Run a local MQTT broker for LAN-based autonomy.
- **More Sensors** - Include further sensors, for example soil temperature, pH or other gas types. You could also add smoke and CO detectors for overall home safety. Use multiple soil moisture sensors for each area for zoned watering.
- **Energy Efficiency** - Incorporate solar panels or battery power for outdoor sensors (pump timer, weather station) to make the system self-sustaining. Implement sleep modes on microcontrollers to save energy when idle.
- **User Interface Upgrades** - Create a custom mobile app or web dashboard (other than Blynk) to view historical data, status of the system, and live video feeds. Add voice feedback (Alexa announcing sensor values), or SMS notifications.
- **Enhanced AI** - Use more optimized image analytics: for example, run person recognition to differentiate family members from strangers, or implement pet detection to stop the water automatically if a pet is in the yard. Get face detection running directly on an edge device for lower latency.
- **Home Integration** - Further integrations for other smart home devices should be added. For example, integrate with a smart lock or a smart thermostat. Use geofencing (phone GPS) or scheduling to automating the lights and watering schedule when the resident comes home. Use weather reports to change watering schedules.
- **Security Enhancements** - Use encryption and authentication protocols to stop unauthorized access for data protection. Communication between the microcontrollers and cloud services is encrypted using SSL/TLS. Logging and alarms must be included for intrusion attempts.

N. Acknowledgements

We would like to thank everyone whose guidance and support helped us in the development of this project.

Firstly, our special thanks would go to **Prof. Bidhan Pramanick**, instructor of our IoT course who played an important role in always guiding, motivating, and supporting us. We strongly appreciate their insight, advice, and feedback that

helped us in the direction and execution of the project Home Automation using IoT.

We would like to acknowledge the Electrical Engineering, IIT Goa and Lab In charge **Mr. Parmeshwar M** and all the TAs for organising the much needed infrastructure, lab equipment, and technical resources required to implement the system.

We are also grateful to our team members, for their constructive suggestions, teamwork, and moral support, without their input, this project would not have progressed as it did.

A special thanks to the open-source communities and documentation of Arduino, Blynk, ESP32 CAM and Sinric Pro that allowed for quick development of IoT projects. We want to acknowledge Google's MediaPipe team for allowing free use of face detection models, and Cloudinary for cloud image storage.

O. References

- Arduino Project Hub (2025). *DIY IoT Plant Watering System using Arduino* <https://projecthub.arduino.cc/>
- Nipanikar, S. I. et al. (2024). *Smart Surveillance System Using ESP32*, *International Journal of Microelectronics and Digital Integrated Circuits* <https://journalspub.com/>
- Sinric Pro [Sinric Pro - Connect Amazon Alexa, Google Home with esp8266, esp32, raspberry pi, RP2040](#)
- Google Developers (2023). *MediaPipe Face Detection* documentation https://mediapipe.readthedocs.io/en/latest/solutions/face_detection.html
- MediaPipe Python [Face detection guide for Python | Google AI Edge | Google AI for Developers](#)
- Blynk Documentation <https://docs.blynk.io/en>
- Flask Documentation <https://flask.palletsprojects.com/en/stable/>
- Cloudinary Documentation <https://cloudinary.com/documentation>
- Dinesh Kumar V. et al. (2024). *Smart Home Automation System*, *International Research Journal on Advanced Engineering Hub (IRJAEH)* <https://irjaeh.com/index.php/>