IoT Enabled Home Security & Automation System

Divyanshu Lakra(2104210) , Gunupuru Tejasri(2104211), Hasitha Varshini(2104235)
Project supervisor: Dr. Bidhan Pramanick
School of Electrical Sciences, IIT Goa, Farmagudi, Goa-403401
bidhan@iitgoa.ac.in

***Abstract*—In today's world, having a home security system is essential as the demand for it is rising. With the daily increase in crime rates, it is crucial to find ways to ensure our safety. While we know there are advanced security systems available, they are often not accessible to everyone. Historically, security has been a significant concern for individuals in companies, homes, and other locations, leaving many people worried about their safety today. A viable solution to these issues could involve integrating door locks with biometric technology and digital locks utilizing passwords or PIN codes. Therefore, our goal is to develop an affordable electronic system capable of recognizing a user's fingerprint and recording images and videos of anyone attempting to bypass the lock. Additionally, in this age of social media, smartphones have become indispensable tools for daily life. The Internet of Things has ushered in a new era that allows us to store contact information, browse the web, send messages, take notes, and manage files and documents effectively. Moreover, we can control devices like lights, fans, and air conditioners. This project is focused on creating home security systems designed to protect specific areas from intruders and detect gas leaks.**

***Keywords*— Arduino mega, Esp8266 module, Raspberry Pi, web camera, R307 Fingerprint sensor, keypad(4x4), Buzzers, switch, temperature & humidity sensor, flame sensor, relays.**

I. Introduction

A security system is characterized by its ability to identify intrusions and unauthorized entries into a building or secured area, preventing such access to safeguard individuals and property from damage or harm. These systems are primarily utilized across residential, commercial, industrial, and military facilities to protect against theft or property destruction, as well as to ensure personal safety from intruders. Vehicle alarms also serve to secure cars and their contents. Furthermore, correctional facilities employ security systems to manage inmates. Among residences, home security systems are particularly prominent. Nowadays, home security and surveillance systems have become vital components of any modern automated household. The fundamental design of a security system starts with assessing the needs of the residents, examining current technologies and equipment, evaluating system costs, considering monitoring options, and ultimately planning for installation.

Biometric authentication methods, including fingerprint and facial recognition technology, have gained popularity due to their dependability and user-friendliness. Fingerprint scanners offer a secure and swift way to allow access, minimizing the chances of unauthorized entry. Likewise, camera-based facial recognition provides an additional security layer by visually confirming identity and notifying users of possible intruders. Research indicates that these systems are more successful at preventing unauthorized access compared to traditional key-based locks or separate alarms.

Keypad security systems, despite being traditional, continue to be a vital component in modern residences because of their straightforwardness and reliability. The combination of keypads with IoT platforms facilitates remote access control, boosting user convenience. For example, users can modify access codes or lock and unlock doors using their smartphones, enhancing both flexibility and security.

Home automation has also emerged as a key aspect of contemporary living. Devices such as 'temperature and humidity sensors" ensure comfortable living conditions, while "flame sensors" offer crucial early warnings against fire dangers. Connecting these sensors to smart applications like the Blynk app allows for real-time oversight and regulation of home devices, such as fans and lights. The ability to manage appliances from a distance promotes energy efficiency and provides homeowners with reassurance.

In spite of these advanced options, current systems still encounter obstacles like high expenses, complicated installation and usability, and insufficient integration. A majority of systems concentrate solely on either security or automation, resulting in a lack of comprehensive solutions.

This project seeks to tackle these issues by creating a compact and effective home security system that integrates "biometric authentication (fingerprint and facial recognition)", "keypad access", and "home automation (temperature, humidity, and flame sensors, alongside appliance management via the Blynk app)". The fusion of these elements into a single, cost-effective solution guarantees user-friendliness while delivering strong security and improved convenience for homeowners.

By utilizing modern technologies, this project presents a comprehensive strategy for home security and automation, aligning with the increasing demand for safe and intelligent living environments.

II. Objectives and Learning Outcomes

1) Enhancing Home Security:
   a) Implement fingerprint recognition, keypad entry, and camera-based identification to establish a multi-tiered security system that guarantees access for only authorized individuals.
   b) Offer real-time surveillance and alerts to notify homeowners of unauthorized entry attempts or any suspicious behavior.
2) Improving Home Safety:
   a) Integrate temperature and humidity detectors to assess environmental factors and maintain optimal comfort conditions.
   b) Deploy a flame detector to identify potential fire threats and send timely warnings to avert emergencies.
   c) Utilize the Blynk app for remote management of household devices, such as lights and fans, through a smartphone.
   d) Allow users to oversee and control their home environment in real-time, promoting convenience and energy savings.

By accomplishing these goals, the project intends to provide a smart and secure living space that tackles contemporary issues in home security and automation while ensuring user convenience, safety, and cost-effectiveness.

III. Methodology

This project was carried out in two main components: Home Security and Home Automation, which aimed at improving safety and enhancing energy efficiency. The following methodology outlines the organized process utilized in the design, execution, and testing of the system.

A. Working of security system:

The security system guarantees regulated access and notifies users in the event of unauthorized entry attempts. The implementation of a fingerprint authentication system utilized the Adafruit Fingerprint Sensor Library, which eased the enrollment and comparison of fingerprints against stored data. When the system detected a recognized fingerprint, it activated the solenoid lock to unlock the door. If the fingerprint wasn't recognized, the user had the option to input a PIN code or password via a keypad. The system permitted a maximum of three tries to enter the correct password. After repeated failures, an alert mechanism was triggered.

To manage unauthorized entry attempts, an intrusion detection feature was created. If a breach occurred, the camera module was triggered using OpenCV, a well-known open-source computer vision framework, which was integrated with the Face-recognition Python library. This setup enabled the system to take images and videos of the intruder, which were subsequently sent to the user via email using the SMTP Library. Moreover, the Pushbullet API was utilized to deliver real-time alerts to the user's smartphone, ensuring they were immediately informed of the issue. The system further included a remote control feature via a smartphone application, allowing users to modify stored fingerprints or reset passwords as needed.

B. Working of automation:

The automation system was designed to facilitate energy-saving measures and environmental monitoring. Lighting management was implemented through the Blynk app, enabling users to control lights remotely, thus fostering efficient energy consumption. Environmental monitoring was conducted using sensors that measured temperature, humidity, and flame levels. The system was designed to notify the user's smartphone in case of unusual sensor readings. These notifications were further enhanced by an audible alarm to capture the user's attention. Pushbullet was employed to provide these alerts in real time.
Testing and validation were essential for confirming the system's reliability. The system underwent comprehensive testing for fingerprint enrollment, password verification, and emergency alert generation. Automation functions, including remote lighting control and environmental monitoring, were evaluated in depth. Notifications and email alerts were checked under simulated breach scenarios to verify the system's efficiency in practical situations.

This methodology showcases the organized strategy adopted to fulfill the project's goals, ensuring a strong and user-friendly home security and automation solution.

IV. Block Diagrams

This flowchart illustrates the combination of home security and automation using an Arduino Mega alongside ESP8266. The security setup features a keypad, a fingerprint scanner, and an LCD for password protection and user feedback, with any breaches setting off an alarm. The automation component utilizes sensors for detecting fires (flame sensor) and monitoring environmental conditions (DHT11), managing devices such as bulbs and fans through the Blynk app. Both systems are linked via the Blynk platform, allowing for remote control, monitoring, and password management to enhance functionality and user convenience.

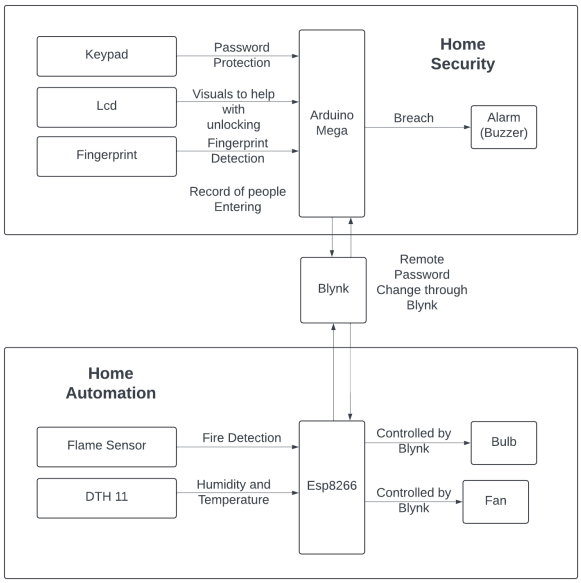


Fig. 1: Block Diagram of home security and automation

V. Circuit Diagram

A. Security system:

The initial diagram illustrates the layout of the security system, which incorporates an Arduino Mega, a 4x4 keypad, an ST7920 LCD display, a fingerprint sensor, an ESP8266 module, a solenoid lock, and a buzzer. The Arduino Mega functions as the main controller, processing input from the keypad and fingerprint sensor to verify users. The LCD offers visual feedback by showcasing messages such as password requests or access status, while the fingerprint sensor enhances security through biometric verification. The ESP8266 module enables wireless communication for remote alerts and app connectivity. The solenoid lock ensures secure physical access, and the buzzer provides audio feedback for occurrences like unsuccessful attempts or granted access.

the figure, does not contain the Raspberry Pi with a camera, as both of them are connected separately and work separately work for the security part.

B. Automation:

The Circuit diagram of the home automation system is illustrated in the second figure, including a DHT11 sensor, a flame sensor, a 4-channel relay module, an ESP8266 module, along with household devices like lights and fans. The DHT11 sensor tracks both temperature and humidity levels, while the flame sensor improves safety by identifying potential fire threats. The relay module manages the operation of connected devices based on data from the sensors or

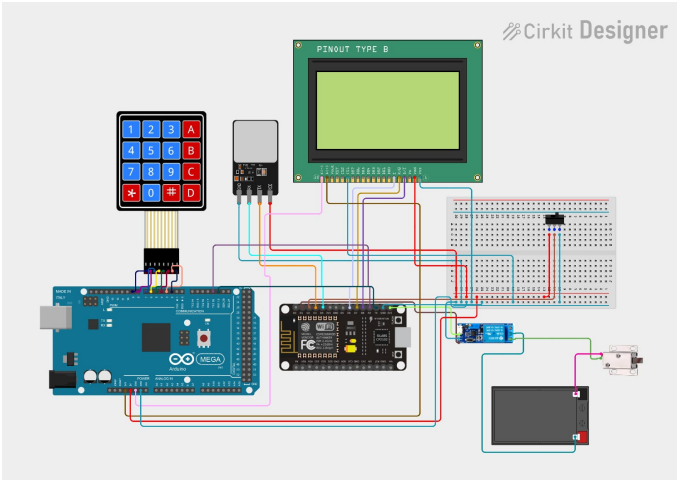


Fig. 2: Circuit diagram of door security system

remote instructions received from the ESP8266 module, which also allows for interaction with a smartphone application for automation and monitoring purposes. A 9V battery powers the entire setup, ensuring that the home automation system operates efficiently and responsively.

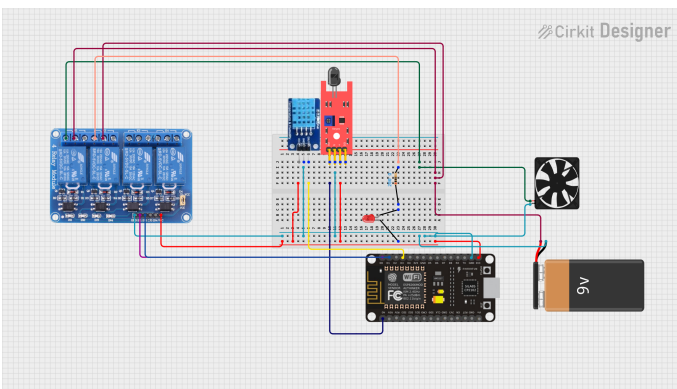


Fig. 3: Circuit diagram of Automation

C. Face Detection:

The Circuit diagram of face detection part is in Figure 3. Raspberry pi is simply connected with webcam which can detect faces using OpenCV library. It further helped us building a Interactive realtime monitoring system for user using Pushbullet

VI. Flowchart

1) **Security System:** This flowchart depicts how a smart home security system functions, starting with the initialization process to load usernames and passwords. Based on the position of the mode switch, the system can operate in either Fingerprint Mode or Keypad Mode. In Fingerprint Mode, users are asked to scan their fingerprint, which is verified before logging the entry and unlocking the door. In Keypad Mode, users must input a password; once verified, access is granted, or if the attempts fail, the system records the failures and activates a Master Unlock Mode after three unsuccessful attempts. Both modes provide secure entry and maintain a record of all activities.

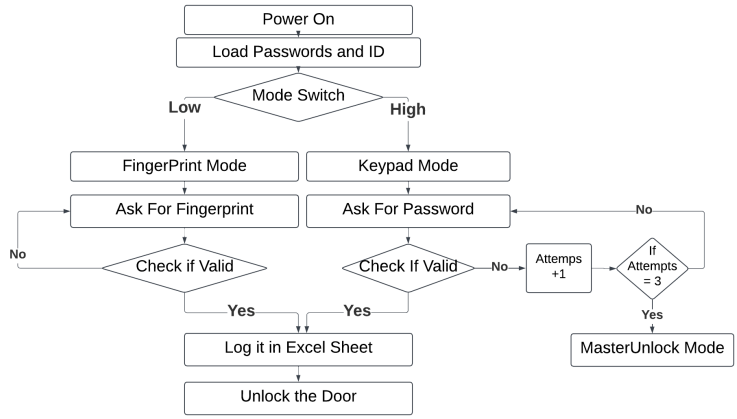


Fig. 4: Flowchart of security system

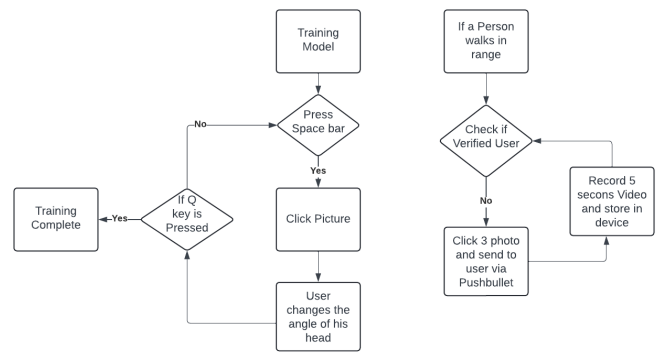


Fig. 5: Flowchart of Face Detection

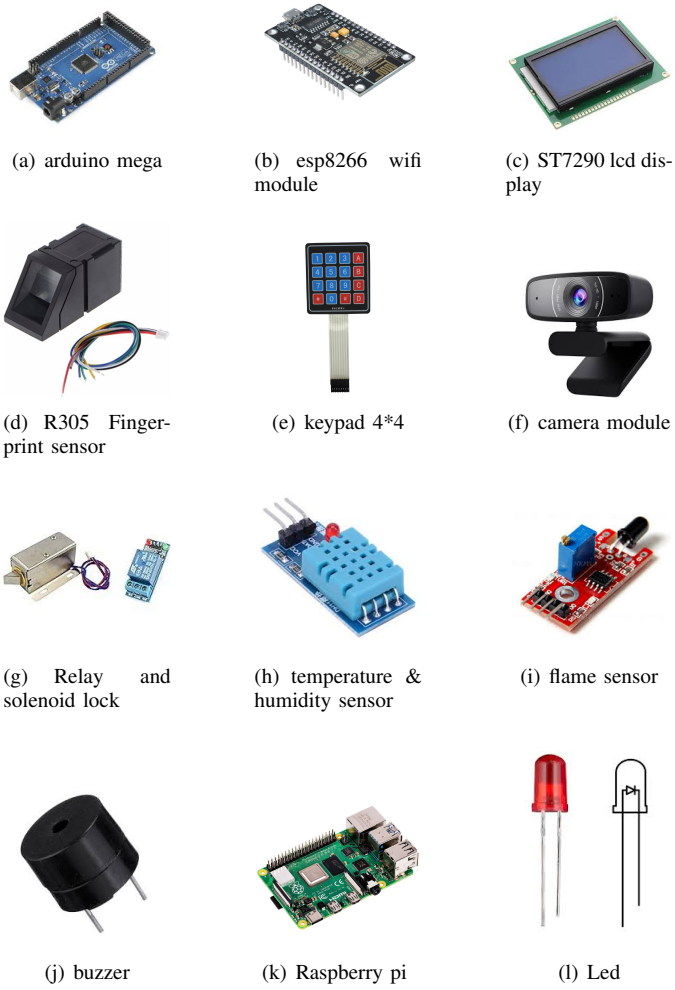
2) **Face detection:** This flowchart depicts how the Face detection code functions, starting with training model with 10 images which is clicked when user presses space bar and the loop is terminated when Q key is pressed in the Keyboard to register user and finish the training Process then the face detection loop which informs user when a unknown face is detected. Which ensures a real time monitoring for user.

VII. Materials/Components

The materials and components employed in this project were meticulously chosen to guarantee the effective implementation of the home security and automation system. Below is a comprehensive breakdown of the hardware and software components used:

A. Hardware components:

1) **Arduino Mega 2560:** This microcontroller unit serves as the core component that manages system operations, including keypad input, and sensor integration.
2) **ESP8266 Wi-Fi Module:** This module enables wireless connectivity between the system (fingerprint sensor and st7920 LCD) and the Blynk application, allowing for notifications and emails to be sent.
3) **ST7290 LCD Display:** A graphic display unit that presents system status updates, including access attempts and environmental data.
4) **Fingerprint sensor:** This component allows for biometric verification, enabling authorized users to gain access by unlocking the door.
5) **Keypad:** A numeric keypad that serves as an alternative means for password-based entry.



(a) arduino mega (b) esp8266 wifi module (c) ST7290 lcd display

(d) R305 Fingerprint sensor (e) keypad 4*4 (f) camera module

(g) Relay and solenoid lock (h) temperature & humidity sensor (i) flame sensor

(j) buzzer (k) Raspberry pi (l) Led

Fig. 6: Hardware components of this project

6) **Camera Module:** Captures photographs and videos during security incidents and sends them for email alerts.
7) **Relay Module and Solenoid Lock:** The relay operates the solenoid lock, which secures the door and activates upon granted access.
8) **Temperature and Humidity sensor:** This sensor tracks environmental conditions and relays data to the system for analysis and alert generation.
9) **Flame sensor:** Detects fire presence and activates an alarm when required.
10) **Buzzer:** Emits a sound in response to unusual conditions, such as unsuccessful access attempts or fire alerts.
11) **LED's:** Function as visual signals indicating system states like lock status and alert conditions.
12) **Power supply:** Supplies the necessary electrical energy to all system components.

B. Software Components:

1) **Adafruit Fingerprint Sensor Library:** Supplies the necessary electrical energy to all system components.
2) **U8g2 Library:** Manages graphical visualization and text display on the ST7920 LCD module.
3) **OpenCV Library:** Provides image and video processing capabilities for facial recognition during security events.
4) **Face recognition Library:** Works alongside OpenCV for accurate facial identification.
5) **SMTP Library:** Enables the system to send email alerts informing users of security incidents.
6) **PushBullet API:** Sends immediate notifications and alerts to the user's smartphone, ensuring efficient communication.
7) **Blynk app:** Offers an intuitive interface for remotely controlling the system, such as toggling lights and monitoring environmental parameters.

These materials and components laid the groundwork for the project, facilitating the integration of security and automation features into a unified system.

VIII. Code developed/Used :

The detailed implementation, including the source code, schematics, and setup instructions, is available on the project's GitHub repository with all the extra helping Scripts and our process also If someone wants to build this from scratch they can follow the instruction given there. Readers can access the full details and source files using the link below:

Home Security and Automation Repository

A. ESP8266 Working Code:

Following code snippets of our Original ESP8266 code contains all the functions which we developed during aur project which contains detecting fingerprint Blynk connection, LCD displaying, Serial Communication between Arduino and ESP8266 for system to work as a unified Block.

1) **Library Setup:**

```
// Blynk credentials
#define BLYNK_TEMPLATE_ID "TMPL3Z2fgiR0X"
#define BLYNK_TEMPLATE_NAME "Home Security"


#include <BlynkSimpleEsp8266.h>
#include <Adafruit_Fingerprint.h>
#include <U8g2lib.h>
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>
#include <ESP8266HTTPClient.h>
#include <SoftwareSerial.h>

char auth[] = "_vV1qCx47WYUvhaoXlyYYlXCzRnoVaMH";
```

2) **Pin and Variables**

```
// Pin Definitions
#define FINGERPRINT_SENSOR_TX 0 // D3
#define FINGERPRINT_SENSOR_RX 2 // D4
#define RELAY_PIN 5
#define MODE_PIN 4
#define MASTER_UNLOCK_PIN 16 // D0

SoftwareSerial mySerial(FINGERPRINT_SENSOR_RX,
    FINGERPRINT_SENSOR_TX);
Adafruit_Fingerprint finger = Adafruit_Fingerprint(&
    mySerial);

// Initialize U8g2 for ST7920 LCD with hardware SPI
U8G2_ST7920_128X64_F_SW_SPI u8g2(U8G2_R0, /* clock=
    */ 12, /* data=*/ 13, /* CS=*/ 15 /* reset=*/ )
    ;

// Wi-Fi credentials
const char* ssid = "Redmi";
const char* password = "72580306";

const char* host = "script.google.com";
const int httpsPort = 443;
WiFiClientSecure client;
String GAS_ID = "
    AKfycbz5ZtGUYEMDNxfhyCmxxSzDiTWvScMR7lW
    NN9YFG5btrLJ9q3dK4QgbvWpTDqIXmShQ";

uint8_t id;
```

```

SoftwareSerial megaSerial(3, 1); // RX, TX for communication with Arduino Mega
```

3) Void Setup

```
void setup() {
    Serial.begin(115200);
    mySerial.begin(57600); // Initialize software serial for fingerprint sensor
    megaSerial.begin(115200);  // Serial communication with Arduino Mega

 // Blynk initialization
    Blynk.begin(auth, ssid, password);

    u8g2.begin();  // Initialize the LCD
    u8g2.clearBuffer();  // Clear the buffer
    u8g2.setFont(u8g2_font_ncenB08_tr);  // Set font

    // Initialize Fingerprint Sensor
    Serial.println("Initializing fingerprint sensor...");
    finger.begin(57600);
    if (finger.verifyPassword()) {
        Serial.println("Fingerprint sensor initialized successfully!");
         u8g2.drawStr(0,10,"Fingerprint Initialized");  // Display the message on the first line
    } else {
        Serial.println("Failed to initialize fingerprint sensor. Check connections.");
        while (1);
    }

    // Initialize LED and relay pins
    pinMode(RELAY_PIN, OUTPUT);
    pinMode(MODE_PIN, INPUT); // Mode selection pin
    digitalWrite(RELAY_PIN, HIGH);
    pinMode(MASTER_UNLOCK_PIN, OUTPUT);
    digitalWrite(MASTER_UNLOCK_PIN, LOW); // Default to LOW

    connectToWiFi(); // Connect to Wi-Fi
}
```

4) Void Loop

```
void loop() {
    Blynk.run(); // Handle Blynk communication
    int mode = digitalRead(MODE_PIN); // Read the state of MODE_PIN (D2)

    if (mode == HIGH) {
        // Fingerprint Unlock Mode
        Serial.println("Fingerprint Unlock Mode");
        detectFingerprint();
    } else {
        // Keypad Unlock Mode
        Serial.println("Keypad Unlock Mode");
        checkKeypadInput();
    }
    delay(500); // Delay before the next loop iteration
}
```

5) Void ConnectToWiFi

```
// Function to connect to Wi-Fi
void connectToWiFi() {
    WiFi.begin(ssid, password);
    Serial.println("Connecting to Wi-Fi...");
    displayMessages("Connecting to Wi-Fi","");
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        delay(500);
    }
    Serial.println("Connected!");
    displayMessages("Connected","");
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
    client.setInsecure();  // Disable SSL verification for the client
}
```

6) Void detectFingerprint

```
// Function to detect fingerprint
void detectFingerprint() {
    Serial.println("\nWaiting for a valid fingerprint...");
    displayMessages("Place the Finger on ","Fingerprint Sensor");
    delay(1000);
    uint8_t id = getFingerprintID();

    if (id != -1 && id != 255) {
        Serial.print("Fingerprint ID: ");
        displayMessages("Valid Fingerprint","Detected");
        Serial.println(id);
        logEntry(id);  // Log the entry to Google Sheets
    } else {
        Serial.println("No valid fingerprint detected or ID is 255.");
        displayMessages("No Valid ID, Try Again","");
    }
}
```

7) Void checkKeypadInput

```
// Function to check keypad input from Arduino
void checkKeypadInput() {
    if (megaSerial.available()) {
        delay(10); // Brief delay
        String message = megaSerial.readStringUntil('\n');  // Read from Arduino Mega
        Serial.println("Message from Mega: " + message);  // Debugging message

        // Split the message into two lines for display
        int delimiterIndex = message.indexOf('|');  // Using '|' as a delimiter
        String line1;
        String line2;

        // Check if the delimiter was found
        if (delimiterIndex != -1) {
            line1 = message.substring(0, delimiterIndex);  // Get the first part
            line2 = message.substring(delimiterIndex + 1);  // Get the second part
        } else {
            line1 = message;  // If no delimiter, use the entire message as line1
            line2 = "";        // Set line2 to empty
        }

        displayMessages(line1, line2);  // Display both lines on the LCD

        // Unlock door if the message indicates so
        if (line1 == "Welcome") {
            logEntry(128);
        }
    }
}
```

8) Void unLockDoor

```
// Function to unlock the door
void unlockDoor() {
    digitalWrite(RELAY_PIN, LOW);  // Unlock the door
    delay(3000);  // Keep the door unlocked for 3 seconds
    digitalWrite(RELAY_PIN, HIGH);  // Lock the door again
    displayMessages("Log Successfull","Door Unlocked");
}
```

9) Void logEntry

```
// Function to log fingerprint entry to Google Sheets
void logEntry(int userId) {
    Serial.println("==========");
```

```
    Serial.print("Connecting to ");
    Serial.println(host);

    if (!client.connect(host, httpsPort)) {
        Serial.println("Connection failed");
        displayMessages("Log Failed, Try Again","");
        return;
    } else {
        Serial.println("Door unlocked");
        unlockDoor();
    }

    String user_ID = String(userId);
    String url = "/macros/s/" + GAS_ID + "/exec?user=" + user_ID;

    client.print(String("GET ") + url + " HTTP/1.1\r\n" +
        "Host: " + host + "\r\n" +
        "User-Agent: BuildFailureDetectorESP8266\r\n" +
        "Connection: close\r\n\r\n");

    Serial.println("Request sent");
    Serial.println("Closing connection");
    Serial.println("==========");
}
```

10) **Void displayMessages**

```
// Function to display messages on LCD
void displayMessages(String line1, String line2) {
    u8g2.clearBuffer();  // Clear the buffer once

    // Get the widths of both messages
    int line1Width = u8g2.getStrWidth(line1.c_str());
    int line2Width = u8g2.getStrWidth(line2.c_str());

    // Calculate the x positions to center the text
    int xPosLine1 = (u8g2.getDisplayWidth() - line1Width) / 2;
    int xPosLine2 = (u8g2.getDisplayWidth() - line2Width) / 2;

    // Draw the centered messages
    u8g2.drawStr(xPosLine1, 20, line1.c_str());  // Display first line
    u8g2.drawStr(xPosLine2, 40, line2.c_str());  // Display second line

    u8g2.sendBuffer();  // Send buffer to display
}
```

11) **Void getFingerprintID**

```
// Function to get Fingerprint ID
uint8_t getFingerprintID() {
    uint8_t p = finger.getImage();
    switch (p) {
        case FINGERPRINT_OK:
            Serial.println("Fingerprint image taken.");
            break;
        case FINGERPRINT_NOFINGER:
            Serial.println("No finger detected.");
            return -1;
        default:
            return -1;
    }

    p = finger.image2Tz();
    if (p == FINGERPRINT_OK) {
        p = finger.fingerFastSearch();
        if (p == FINGERPRINT_OK) {
            return finger.fingerID;
        }
    }
    return -1;
}
```

12) **Void BlynkWrite**

```
// Blynk function to monitor V0
BLYNK_WRITE(V0) {
    int pinValue = param.asInt(); // Get the value of V0
    if (pinValue == 1) {
        digitalWrite(MASTER_UNLOCK_PIN, HIGH); // Signal Arduino to exit Master Unlock
        delay(500);
        digitalWrite(MASTER_UNLOCK_PIN, LOW); // Signal Arduino to exit Master Unlock

    } else {
        digitalWrite(MASTER_UNLOCK_PIN, LOW);
    }
}
```

B. *Arduino Working Code:*

Arduino code Handles Keypad Mode, users must input a password; once verified, access is granted, or if the attempts fail, the system records the failures and activates a Master Unlock Mode after three unsuccessful attempts. It also allows user to set password using Blynk to achieve versatility and secure system.

1) **Library and pins Setup:**

```
#include <Keypad.h>
#include <SoftwareSerial.h>
#include <EEPROM.h>  // Include EEPROM library

// Keypad configuration
const byte ROWS = 4;  // Four rows
const byte COLS = 4;  // Four columns
char keys[ROWS][COLS] = {
  { '1', '2', '3', 'A' },
  { '4', '5', '6', 'B' },
  { '7', '8', '9', 'C' },
  { '*', '0', '#', 'D' }
};

byte rowPins[ROWS] = { 47, 45, 43, 41 };  // Row pin connections
byte colPins[COLS] = { 39, 37, 35, 33 };  // Column pin connections

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
```

2) **Variables**

```
// Password management variables
String newPassword = "";
String enteredPassword = "";
const byte maxPasswordLength = 6;
bool isPasswordSet = false;
bool isMasterUnlockMode = false;  // New variable to track Master Unlock mode
byte attemptCount = 0;
const int masterUnlockPin = 31;  // Pin number for the signal from ESP (D0)


// Serial communication
SoftwareSerial espSerial(18, 19);  // RX, TX for ESP8266
const int passwordAddress = 0;     // EEPROM address for password storage

// Switch configuration
const int switchPin = 27;          // Pin number for the switch
bool previousSwitchState = HIGH;  // Track the previous state of the switch
```

3) Void Setup

```
void setup() {
  Serial.begin(115200);      // Serial communication with computer
  espSerial.begin(115200);  // Communication with ESP8266

  // Setup pin modes
  pinMode(masterUnlockPin, INPUT);  // Set the master unlock pin as input
  pinMode(switchPin, INPUT);

  loadPasswordFromEEPROM();  // Load password from EEPROM on startup

  if (isPasswordSet) {
    espSerial.println("Password Loaded. | Enter Password");
  } else {
    espSerial.println("Set New Password");  // Prompt to set a new password
  }
}
```

4) Void Loop

```
void loop() {

  // Check the switch state
  bool currentSwitchState = digitalRead(switchPin);

  // fingerprint mode to keypad mode
  if (currentSwitchState == LOW && previousSwitchState == HIGH) {
    resetEnteredPassword();
    if (isPasswordSet) {
      espSerial.println("Enter Password");
    }
  }
  // Updateprevious state
  previousSwitchState = currentSwitchState;

  // Only process keypad input if in keypad mode (switch is low)
  if (currentSwitchState == LOW) {
    char key = keypad.getKey();  // Get the pressed key from the keypad

    if (key) {
      if (!isPasswordSet) {
        handlePasswordSetting(key);  // Handle password setting process
      } else if (!isMasterUnlockMode) {
        handlePasswordEntry(key);  // Handle password entry
      }
    }
  }
}
```

5) Void handlePasswordEntry

```
// Function to handle password entry
void handlePasswordEntry(char key) {
  if (((key >= '0' && key <= '9') || (key >= 'A' && key <= 'D')) && enteredPassword.length() < maxPasswordLength) {
    enteredPassword += key;  // Add key to entered password
    espSerial.print("Entered Password: |");
    espSerial.println(enteredPassword);  // Send entered password to ESP8266 for display
  }

  if (key == '#') {
    checkPassword();  // Check if entered password is correct
  }

  // Reset entered password if '*' is pressed
  if (key == '*') {
    resetEnteredPassword();
  }
}
```

6) Void handlePasswordSetting

```
// Function to handle setting a new password
void handlePasswordSetting(char key) {
  espSerial.println("Set New Password: |");
  if (((key >= '0' && key <= '9') || (key >= 'A' && key <= 'D')) && newPassword.length() < maxPasswordLength) {
    newPassword += key;  // Add key to new password
    espSerial.print("New Password: |");
    espSerial.println(newPassword);  // Send new password to ESP8266 for display
  }

  if (key == '#') {  // Confirm password
    isPasswordSet = true;
    savePasswordToEEPROM(newPassword);  // Save the password to EEPROM
    espSerial.println("Password Set | Enter Password");
    newPassword.trim();  // Remove leading and trailing whitespace
  }
}

// Function to handle password entry
void handlePasswordEntry(char key) {
  if (((key >= '0' && key <= '9') || (key >= 'A' && key <= 'D')) && enteredPassword.length() < maxPasswordLength) {
    enteredPassword += key;  // Add key to entered password
    espSerial.print("Entered Password: |");
    espSerial.println(enteredPassword);  // Send entered password to ESP8266 for display
```

7) Void handleWrongPassword

```
// Function to handle wrong password attempts
void handleWrongPassword() {
  attemptCount++;

  if (attemptCount == 1) {
    espSerial.println("Wrong Password, | 2 chances left.");
    resetEnteredPassword();
    delay(500);
    espSerial.println("Enter Password.");

  } else if (attemptCount == 2) {
    espSerial.println("Wrong Password, | 1 chance left.");
    resetEnteredPassword();
    delay(500);
    espSerial.println("Enter Password.");

  } else if (attemptCount >= 3) {
    enterMasterUnlockMode();  // Enter Master Unlock mode after 3 failed attempts
  }
}
```

8) Void checkPassword

```
// Function to check the entered password
void checkPassword() {
  if (enteredPassword == newPassword) {
    espSerial.println("Correct Password!");
    delay(500);
    espSerial.println("Welcome|Door Unlocked");
    delay(3000);              // Wait for 3 seconds before resetting
    resetEnteredPassword();  // Reset entered password
    attemptCount = 0;         // Reset attempt count
  } else {
    handleWrongPassword();  // Handle wrong password attempt
  }
}
```

9) Void enterMasterUnlockMode

```
// Function to enter Master Unlock mode
void enterMasterUnlockMode() {
  isMasterUnlockMode = true;  // Set flag to indicate Master Unlock mode
  espSerial.println("No more tries|MasterLocked");

 // Prompt the user to enter '000' to reset the password or check for the master unlock signal from pin 31
while (isMasterUnlockMode) {
    espSerial.println("Master Unlock | Required"); // Prompt message

    // Check if the signal on masterUnlockPin (pin 31) is HIGH
    int signal = digitalRead(masterUnlockPin);
    if (signal == HIGH) {
        espSerial.println("MasterUnlock Triggered| by Blynk"); // Indicate it was triggered by pin
        resetPassword();                                    // Reset the password
        espSerial.println("Set New Password: |");  // Prompt to set a new password
        isMasterUnlockMode = false;                         // Exit Master Unlock mode
        break;                                              // Break out of the loop
    }

    // Check for '000' input from the keypad
    if (checkResetPassword()) {
        espSerial.println("MasterUnlock Triggered| by Input '000'"); // Indicate it was triggered by keypad input
        resetPassword();                                    // Reset password
        espSerial.println("Set New Password: |");  // Prompt to set a new password
        isMasterUnlockMode = false;                         // Exit Master Unlock mode
    }
}
}
```

10) Void exitMasterUnlockMode

```
// Function to exit Master Unlock mode
void exitMasterUnlockMode() {
  isMasterUnlockMode = false;
  espSerial.println("Master Unlock Mode Exited");
  attemptCount = 0;  // Reset attempt count
  resetPassword();
}
```

11) Void checkResetPassword

```
// Function to check for '000' entry to reset the password
bool checkResetPassword() {
  String zeros = "";
  for (int i = 0; i < 3; i++) {
    char key = keypad.getKey();  // Use getKey() for non-blocking checks
    while (!key) {               // Loop until a key is pressed
      key = keypad.getKey();
    }

    if (key == '0') {
      zeros += key;  // Add zero to the string
    } else {
      return false;  // If any key is not '0', exit
    }
  }
  return (zeros == "000");  // Return true if '000' was entered
}
```

12) Void resetPassword

```
// Function to reset the password
void resetPassword() {
  newPassword = "";         // Clear the new password
  isPasswordSet = false;    // Mark the password as not set
  attemptCount = 0;         // Reset the attempt count
  resetEnteredPassword();   // Clear the entered password for security
}
```

13) Void resetEnteredPassword

```
// Function to reset the entered password
void resetEnteredPassword() {
  enteredPassword = "";  // Clear entered password
}
```

14) Void savePasswordToEEPROM

```
// Function to save the password to EEPROM
void savePasswordToEEPROM(String password) {
  for (int i = 0; i < maxPasswordLength; i++) {
    if (i < password.length()) {
      EEPROM.write(passwordAddress + i, password[i]);  // Write each character of the password
    } else {
      EEPROM.write(passwordAddress + i, '\0');  // Null terminate if password is shorter
    }
  }
}
```

15) Void savePasswordToEEPROM

```
// Function to load the password from EEPROM
void loadPasswordFromEEPROM() {
  char password[maxPasswordLength + 1];  // +1 for null terminator
  for (int i = 0; i < maxPasswordLength; i++) {
    password[i] = EEPROM.read(passwordAddress + i);
  }
  password[maxPasswordLength] = '\0';          // Null terminate the string
  newPassword = String(password);              // Assign read password to newPassword
  isPasswordSet = (newPassword.length() > 0);  // Set isPasswordSet flag based on read password
}
```

C. Raspberry pi Code

1) Initialization

```
import cv2
import imutils
import pickle
import time
from datetime import datetime
from pathlib import Path
from imutils.video import VideoStream
from pushbullet import Pushbullet
import face_recognition
import threading

# Initialize necessary variables
currentname = "unknown"
last_detection_time = 0
detection_cooldown = 30  # Cooldown time in seconds
encodingsP = "encodings.pickle"
cascade = "haarcascade_frontalface_default.xml"
ACCESS_TOKEN = "o.SA5FAH5SS1gFrXfKvlpBn24MCbtwF0Gm"

# Initialize Pushbullet object
pb = Pushbullet(ACCESS_TOKEN)

# Load known faces and embeddings
print("[INFO] Loading encodings and face detector")
try:
    data = pickle.loads(open(encodingsP, "rb").read())
except FileNotFoundError:
```

```python
    print(f"[ERROR] Encodings file '{encodingsP}' not found. Exiting...")
    exit(1)

detector = cv2.CascadeClassifier(cascade)
```

2) Video Stream Initialization

```python
# Start the video stream
print("[INFO] Starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)
```

3) Utility Functions

- **Pushbullet Notifications**

```python
def send_pushbullet_notification(image_paths):
    try:
        # Send a text alert
        pb.push_note("Security Alert", "Unknown person detected! ")

        # Attach all images
        for image_path in image_paths:
            with open(image_path, "rb") as pic:
                file_data = pb.upload_file(pic, image_path.name)
            pb.push_file(**file_data)

        print("[ALERT] Intruder images sent!")
    except Exception as e:
        print(f"Failed to send notification: {e}")

def send_pushbullet_notification_async(image_paths):
    notification_thread = threading.Thread(target=send_pushbullet_notification, args=(image_paths,))
    notification_thread.start()
```

- **Record Video**

```python
def record_video(output_folder, frame_width, frame_height):
    print("[INFO] Starting 5-second video recording...")
    video_path = output_folder / "recording.avi"
    out = cv2.VideoWriter(
        str(video_path),
        cv2.VideoWriter_fourcc(*'MJPG'),
        20.0,
        (frame_width, frame_height)
    )

    start_time = time.time()
    while time.time() - start_time < 5:  # Record for 5 seconds
        frame = vs.read()

        # Safety check for NoneType frame
        if frame is None:
            print("[ERROR] Could not capture frame during video recording.")
            break

        frame = imutils.resize(frame, width=frame_width)
        out.write(frame)

    out.release()
    print(f"[INFO] Video saved to {video_path}")
```

4) Create an Output Folder

```python
def create_output_folder():
    timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
    folder_path = Path(f"./IntruderLogs/{timestamp}")
    folder_path.mkdir(parents=True, exist_ok=True)
    return folder_path
```

5) Main Detection Loop

```python
frame_counter = 0
FRAME_SKIP = 10  # Number of frames to skip (process every 10th frame)

while True:
    frame = vs.read()

    # Safety check for NoneType frame
    if frame is None:
        print("[ERROR] Could not capture frame from video stream. Exiting...")
        break

    # Increment frame counter
    frame_counter += 1

    # Process only every 10th frame
    if frame_counter % FRAME_SKIP != 0:
        continue

    # Reduce frame resolution for lower memory and CPU usage
    frame = imutils.resize(frame, width=320)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    rects = detector.detectMultiScale(
        gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30), flags=cv2.CASCADE_SCALE_IMAGE)
    boxes = [(y, x + w, y + h, x) for (x, y, w, h) in rects]

    encodings = face_recognition.face_encodings(rgb, boxes)
    names = []

    for encoding in encodings:
        matches = face_recognition.compare_faces(data["encodings"], encoding)
        name = "Unknown"

        if True in matches:
            matchedIdxs = [i for (i, b) in enumerate(matches) if b]
            counts = {}
            for i in matchedIdxs:
                name = data["names"][i]
                counts[name] = counts.get(name, 0) + 1
            name = max(counts, key=counts.get)

        if name == "Unknown" and (time.time() - last_detection_time > detection_cooldown):
            last_detection_time = time.time()
            print("[ALERT] Unknown person detected!")

            output_folder = create_output_folder()
            print(f"[INFO] Created folder: {output_folder}")

            # Capture 3 images at 1-second intervals
            captured_images = []
            for i in range(3):
                img_path = output_folder / f"captured_image_{i + 1}.jpg"
                cv2.imwrite(str(img_path), frame)
                captured_images.append(img_path)
                time.sleep(1)

            send_pushbullet_notification_async(captured_images)

            frame_width = frame.shape[1]
            frame_height = frame.shape[0]
            video_thread = threading.Thread(target=record_video, args=(output_folder, frame_width, frame_height))
            video_thread.start()

        names.append(name)
```

6) **Cleanup**
Finally, cleanup resources when the program exits.

```
print("[INFO] Cleaning up...")
cv2.destroyAllWindows()
vs.stop()
```

7) **Script for Training Model**

```
#! /usr/bin/python

# import the necessary packages
from imutils import paths
import face_recognition
#import argparse
import pickle
import cv2
import os

# our images are located in the dataset folder
print("[INFO] start processing faces...")
imagePaths = list(paths.list_images("dataset"))

# initialize the list of known encodings and known names
knownEncodings = []
knownNames = []

# loop over the image paths
for (i, imagePath) in enumerate(imagePaths):
  # extract the person name from the image path
  print("[INFO] processing image {}/{}".format(i + 1,
    len(imagePaths)))
  name = imagePath.split(os.path.sep)[-2]

  # load the input image and convert it from RGB (OpenCV ordering)
  # to dlib ordering (RGB)
  image = cv2.imread(imagePath)
  rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

  # detect the (x, y)-coordinates of the bounding boxes
  # corresponding to each face in the input image
  boxes = face_recognition.face_locations(rgb,
    model="hog")

  # compute the facial embedding for the face
  encodings = face_recognition.face_encodings(rgb, boxes)

  # loop over the encodings
  for encoding in encodings:
    # add each encoding + name to our set of known names and
    # encodings
    knownEncodings.append(encoding)
    knownNames.append(name)

# dump the facial encodings + names to disk
print("[INFO] serializing encodings...")
data = {"encodings": knownEncodings, "names": knownNames}
f = open("encodings.pickle", "wb")
f.write(pickle.dumps(data))
f.close()
```

This Python script encodes faces from a dataset for recognition using the face recognition library. It processes images stored in subdirectories named after individuals, extracting names from folder paths. Each image is converted to RGB, and faces are detected using the HOG model. For detected faces, the script generates 128-dimensional facial embeddings (encodings). These encodings, along with corresponding names, are saved in encodings.pickle for later use. This serialized file enables efficient face recognition by matching new images against the stored encodings. The program is ideal for preparing datasets for applications like access control and surveillance systems.

8) **Script for Face registration**

```
import cv2

name = 'Tejasri' #replace with your name

cam = cv2.VideoCapture(0)

cv2.namedWindow("press space to take a photo", cv2.WINDOW_NORMAL)
cv2.resizeWindow("press space to take a photo", 500, 300)

img_counter = 0

while True:
    ret, frame = cam.read()
    if not ret:
        print("failed to grab frame")
        break
    cv2.imshow("press space to take a photo", frame)

    k = cv2.waitKey(1)
    if k%256 == 27:
        # ESC pressed
        print("Escape hit, closing...")
        break
    elif k%256 == 32:
        # SPACE pressed
        img_name = "dataset/"+ name +"/image_{}.jpg".format(img_counter)
        cv2.imwrite(img_name, frame)
        print("{} written!".format(img_name))
        img_counter += 1

cam.release()

cv2.destroyAllWindows()
```

This Python script captures photos using a webcam and saves them in a structured dataset for face recognition. Users specify their name, which determines the folder where the images are stored (e.g., dataset/Tejasri/). The webcam feed is displayed in a resizable window, allowing users to preview the live video. Press the spacebar to capture and save images with sequential filenames (image0.jpg, image1.jpg, etc.). Pressing ESC exits the program. Captured images are organized in subfolders named after the individual, making the dataset ready for use in face recognition applications or similar projects.

D. *Automation code*

This code integrates a NodeMCU ESP8266 with DHT11 and flame sensors to monitor and control devices using the Blynk IoT platform. Below is a breakdown of its functionality:

1) **Setup Function** The setup function initializes the hardware and establishes a connection to Blynk.

```
void setup() {
  Serial.begin(9600);

  pinMode(Switch_1, OUTPUT);
  pinMode(Switch_2, OUTPUT);
  pinMode(FLAME_PIN, INPUT);
  digitalWrite(Switch_1, 0);
  digitalWrite(Switch_2, 0);

  Blynk.begin(auth, ssid, pass, "blynk.cloud", 80);

  dht.begin();

  // Setup a function to be called every second
  timer.setInterval(1000L, sendSensor);
}
```

2) **Flame Sensor Function** The flame sensor function detects the presence of a flame and triggers a buzzer if necessary.

```
void flameSensor() {
  int flameThreshold = 250;
  int flame_value = digitalRead(FLAME_PIN);
  Serial.println(flame_value);

  Blynk.virtualWrite(V2, flame_value);

  if (flame_value < flameThreshold) {
    digitalWrite(Buzzer, HIGH);
    Serial.println("No flame => No fire detected");
  } else {
    Serial.println("Flame present => Fire detected");
    digitalWrite(Buzzer, LOW);
  }
}
```

3) **DHT11 Sensor Data** The ‘sendSensor‘ function reads temperature and humidity values from the DHT11 sensor and sends them to Blynk virtual pins.

```
void sendSensor() {
  float h = dht.readHumidity();
  float t = dht.readTemperature();  // or dht.readTemperature(true) for Fahrenheit

  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  Blynk.virtualWrite(V1, t); // Temperature
  Blynk.virtualWrite(V0, h); // Humidity
}
```

4) **5. Switch Control Functions** The ‘BLYNK WRITE‘ functions control switches based on values received from the Blynk app.

```
BLYNK_WRITE(V3) {
  int Switch1 = param.asInt();
  digitalWrite(Switch_1, Switch1);
}

BLYNK_WRITE(V4) {
  int Switch2 = param.asInt();
  digitalWrite(Switch_2, Switch2);
}

BLYNK_WRITE(V5) {
  int Switch3 = param.asInt();
  digitalWrite(Switch_3, Switch3);
}

BLYNK_WRITE(V6) {
  int Switch4 = param.asInt();
  digitalWrite(Switch_4, Switch4);
}
```

5) **void Loop** The ‘loop‘ function ensures Blynk and timers are continuously running, and sensor functions are called periodically.

```
void loop() {
  Blynk.run();
  timer.run();
  sendSensor();
  flameSensor();
}
```

IX. 3D-Models

A. *Security system Box design*

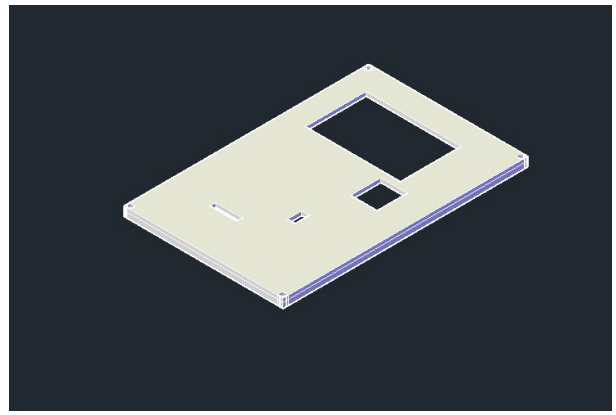


Fig. 7: 3D model of the Cover for the Security System Box, representing the arrangement of input and output openings for convenient access to the security components.

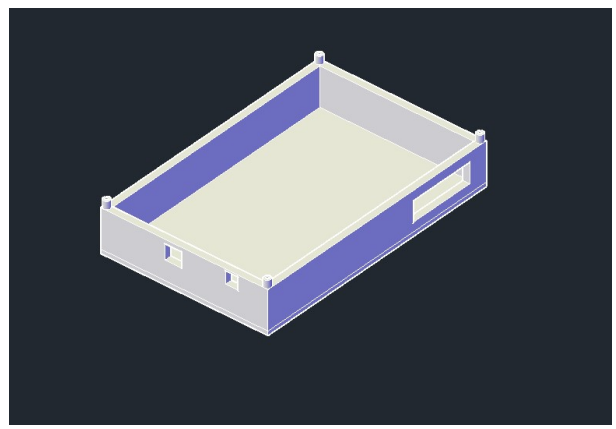


Fig. 8: 3D model of the Base for the Security System Box, depicting the primary housing structure along with openings for attaching and connecting components such as the fingerprint scanner and keypad.

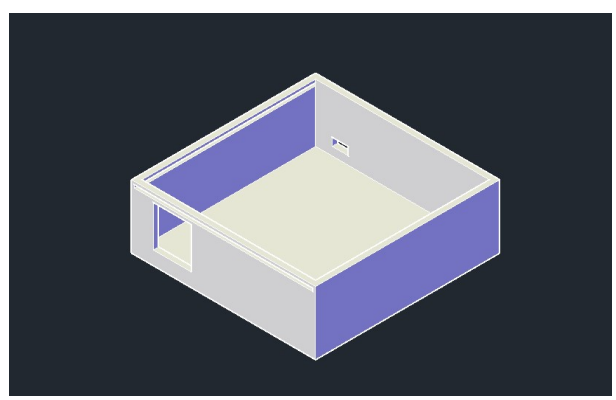


Fig. 9: 3D model of the Base of the solenoid lock box, which primarily contains of relay and battery with the lock

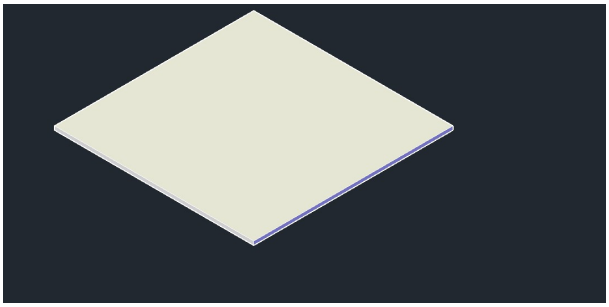


Fig. 10: A 3D model of the lock Box Cover, crafted to shield and secure the internal components while ensuring adequate ventilation and cable management.

B. Automation box design

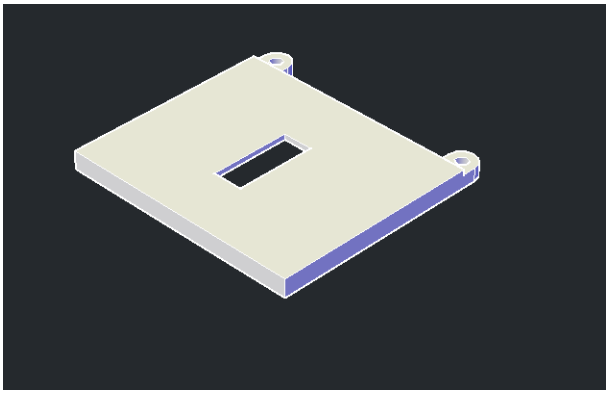


Fig. 11: A 3D model of the Automation Box Cover, crafted to shield and secure the internal components.top opening for the flame and temperature sensors to measure the surrounding units.

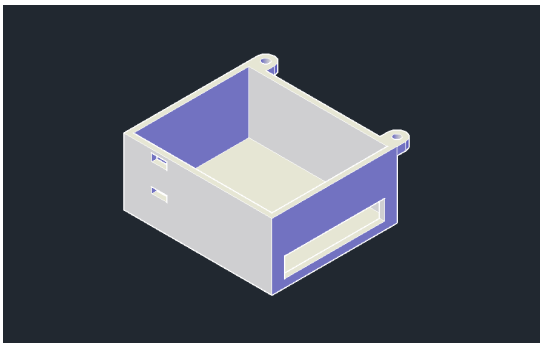


Fig. 12: A 3D model of the Automation Box Base, which includes compartments for the installation of automation control circuits, such as the ESP8266 and relay modules.

X. Results and Discussion

A. Security part:

1) **Results of fingerprint and keypad password**

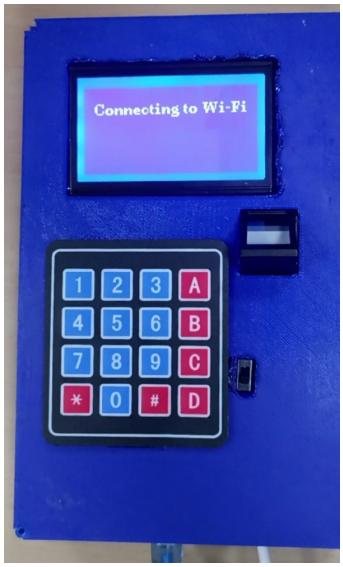


Fig. 13: Connecting to WiFi.

Fig. 14: Entering fingerprint to recognize

Fig. 15: It got detected by the device

Fig. 16: It unlocked the door

Fig. 19: Now, by switching the switch from low to high, we can use the password code to open the door

Fig. 17: Logged in Google Sheet successfully

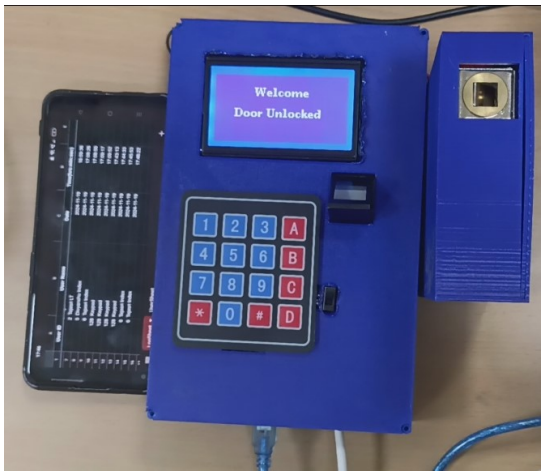


Fig. 20: By entering the correct password, the door gets unlocked

Fig. 18: it just enters the login details in the sheet n

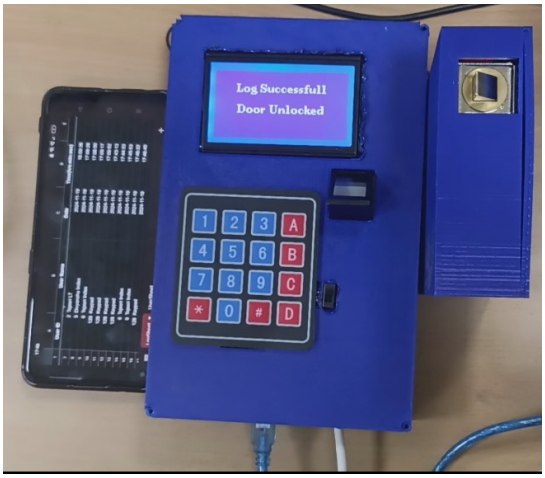


Fig. 21: log in detail in excel sheet as entering by keypad password.

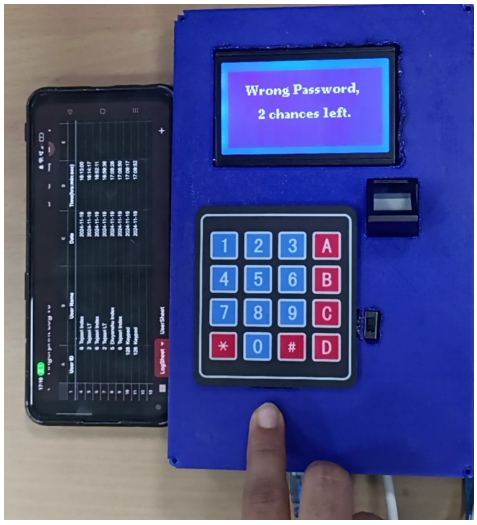
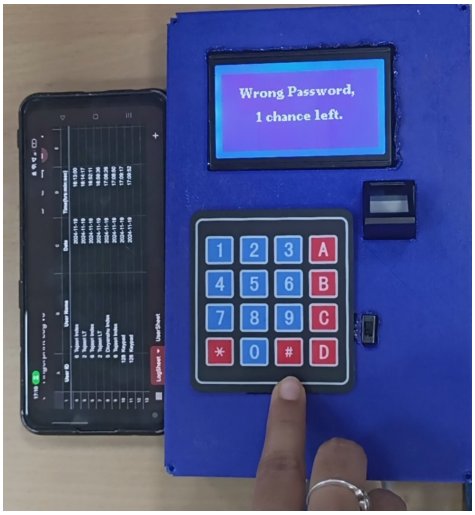


Fig. 22: If in case, we enter the wrong password. we get 2 more chances to enter the correct one



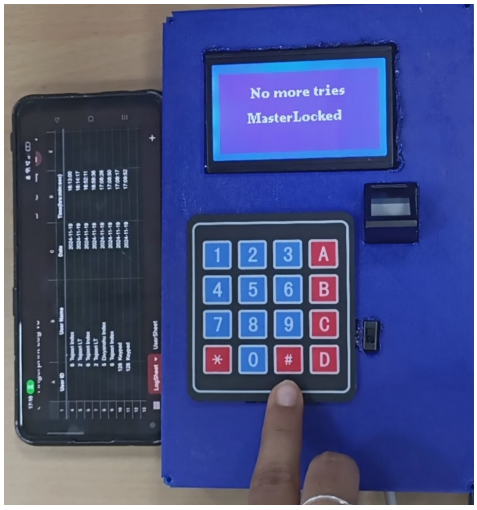


Fig. 23: After 3 Wrong Attempts, no more chances to enter.

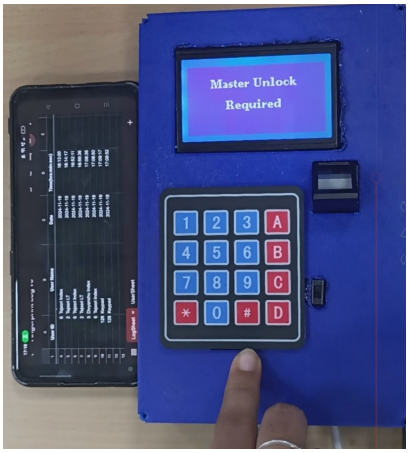


Fig. 24: It went into master unlock mode. Owner input is required to set a new password.

.

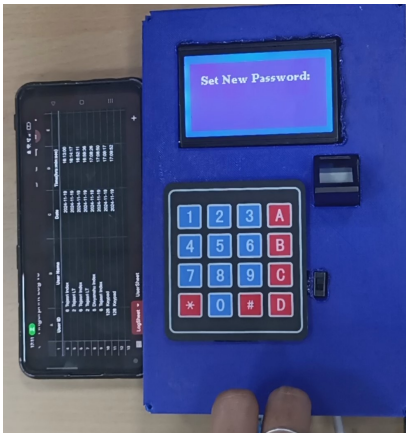


Fig. 25: After Getting owner input, we can set a new password and again entering the password we can unlock the lock

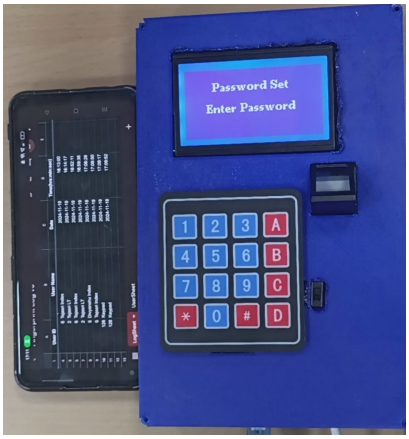


Fig. 26: We set the New Password

2) **Results of camera recognition**

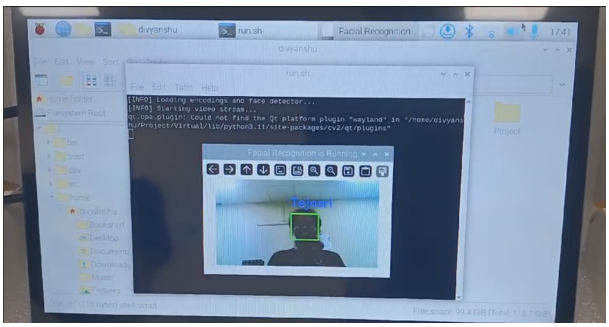


Fig. 27: Recognition of known person

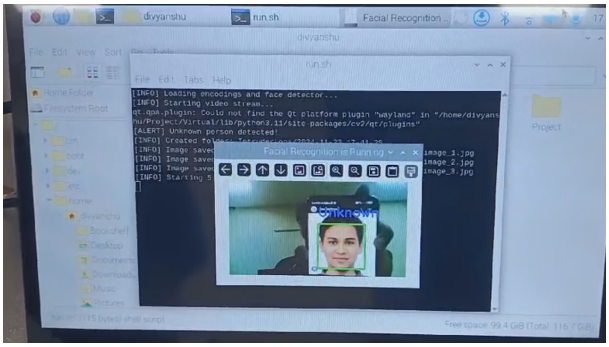


Fig. 28: Recognition of unknown person

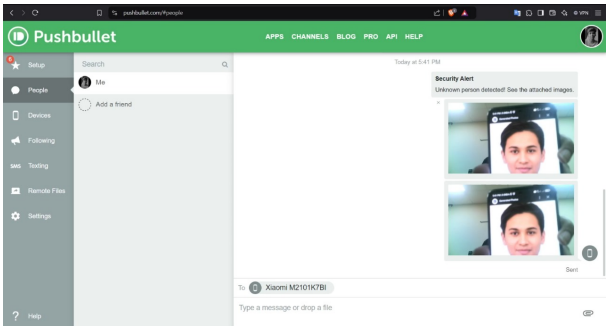
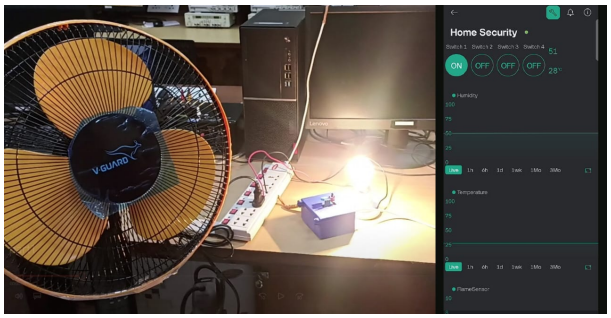


Fig. 29: Pushbullet alert notification by detecting unknown person

B. Automation part



4

Fig. 30: When we operate the device with Bylnk app, only the light switch is on and a fan is off.

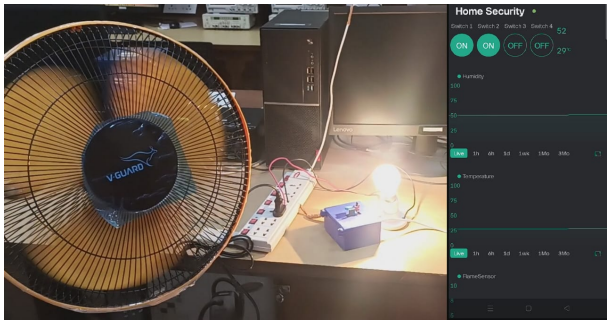


Fig. 31: When both are on.

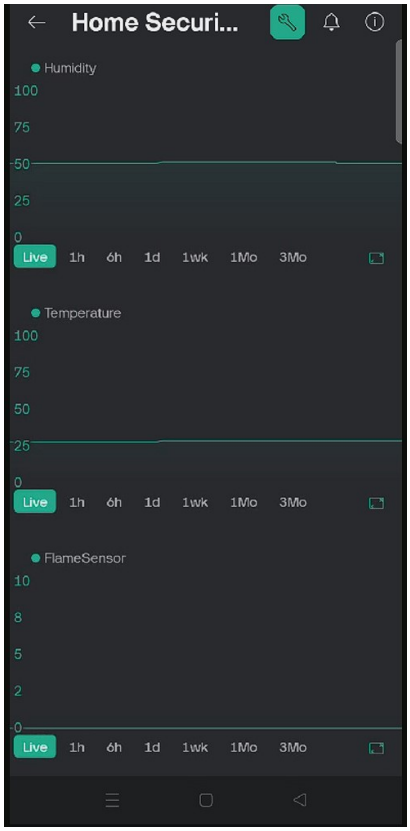


Fig. 32: We can observe the temperature and humidity values in a room and if any flames are not.

Link to All the Demo Videos for the Project:
Demo Videos of Home Security and Automation .

Results: The constructed circuit effectively showcased the desired functionality, facilitating secure access control and streamlined home automation. The system is intended for practical use, with security elements like fingerprint and keypad modules located outside the entrance to manage access. This arrangement guarantees strong access control for improved security.

Within the door, further features such as surveillance and energy-saving solutions were integrated to enhance the management of electrical devices. By automating the operation of lights and fans based on occupancy and user preferences, the system encourages energy conservation within the home.

These outcomes confirm the practicality of installing the system in real-world settings, offering both security and smart home automation capabilities.

Drawbacks: Originally, the project aimed to utilize only a fingerprint sensor and a keypad for security functionalities. However, during the development phase, the addition of face detection was suggested as a supplementary feature. By that stage, the circuit design had already been completed, and a full transition to a Raspberry Pi for managing all operations would have necessitated a significant redesign, which included the removal of ESP8266 and Arduino from the configuration. Given the time limitations and the complexity involved in reconfiguring the system, the face detection module was implemented as a separate element rather than being fully integrated with the primary system.

Furthermore, the ESP8266 microcontroller encountered hardware restrictions, as all available pins had been allocated to the security subsystem. This limitation further complicated the integration of the face detection module and the connections for automation.

In addition, the automation component, designed to connect manual and remote-controlled (Blynk) switches for lights and fans, could not be completely implemented. The ESP8266 did not possess enough resources to effectively support both the security subsystem and the necessary automation functionalities. To enhance automation capabilities, a more powerful microcontroller or supplementary modules would be needed to manage the added complexity.

XI. Summary

The "Smart Home Security and Automation System" represents a contemporary and innovative method of merging home security with automation, delivering increased safety, convenience, and efficiency. This system integrates both biometric and password-based access control alongside automated monitoring and management of household devices, ensuring a smooth user experience. This project presents a budget-friendly, simple-to-install solution that boosts security and automates everyday tasks, minimizing the need for human interaction and enhancing overall safety. The implementation was successful, fulfilling design requirements and showcasing its potential for real-world applications in both residential and commercial environments. This system signifies a noteworthy advancement in modern home technology, providing a secure and automated way of living.

XII. Future works

In the future works section, the following enhancements and developments are planned to further improve the system:

1) **Further Development in Branches with specific function:**As this was a Prototype build keeping in mind the many functionalities which we implemented. We can take it a Step further by Modularising it in Different Products for specific needs like only keypad and Fingerprint lock with no IoT for people who don't need IoT, then we can make a branch where all of them are integrated but with cost reduction and only on microcontroller as its head.
2) **Battery Backup for Security Component:**A battery backup will be added to the security component to guarantee continuous operation during power failures or in areas without consistent electricity supply. This will boost the system's dependability and self-sufficiency.
3) **LCD Screen for Video Feed :** Future plans include the addition of an LCD screen to display live camera feeds or video streaming directly on the system's interface. This will allow users to monitor the security condition of their premises in real time.
4) **Enhanced Mobile App Design:** There will be a thorough redesign of the mobile application to enhance the user interface and overall user experience. This will involve improved navigation, features for managing the automation system, and capabilities for receiving notifications and alerts from the security setup.
5) **Reduction of Microcontroller Usage:** The existing system employs several microcontrollers for various functions. The objective is to streamline the setup by minimizing the number of microcontrollers required, thus simplifying the system and enhancing efficiency, cost-effectiveness, and maintenance ease.
6) **Integration of PIR Motion Sensor in Automation:** A Passive Infrared (PIR) motion sensor will be incorporated into the automation system to facilitate automatic responses to human movement or presence. This will improve security and enable actions like activating lights, fans, or alarms when motion is detected.
7) **Interlinking Manual and App Controls:** The system will undergo improvements to enable seamless switching between manual and app-based control. This will provide users the flexibility to operate the automation system via the app or through manual controls, ensuring ease of use in any situation. These upcoming enhancements will enhance the scalability and adaptability of the system, making it more efficient, user-friendly, and resilient.

XIII. Acknowledgment

We are thrilled to convey our heartfelt appreciation to our project coordinator, [Dr. Bidhan Pramanick], for their essential guidance, encouragement, and steadfast support throughout our project, "Home Automation and Security System." Their insightful feedback and expert recommendations played a crucial role in the successful completion of this project.
Furthermore, we wish to recognize all the professors and lecturers who have guided us during our studies, equipping us with the knowledge and resources necessary to carry out this project.

Finally, we would like to express our gratitude to our friends and colleagues who helped us throughout the project, providing their collaboration and encouragement. This project would not have been achievable without their support.

References

[1] Newbiely, "ESP8266 DHT11," Newbiely. [Online]. Available: https://newbiely.com/tutorials/esp8266/esp8266-dht11.
[2] Electronics Lab, "Home Automation Using NodeMCU ESP8266 Board," Electronics Lab. [Online]. Available: https://www.electronics-lab.com/project/home-automation-using-nodemcu-esp8266-board.
[3] Newbiely, "ESP8266 Flame Sensor," Newbiely. [Online]. Available: https://newbiely.com/tutorials/esp8266/esp8266-flame-sensor.
[4] SurtrTech, "Keypad Door Lock with Changeable Code," Arduino Project Hub. [Online]. Available:https://projecthub.arduino.cc/SurtrTech/keypad-door-lock-with-changeable-code-fb3de2.
[5] R.B, "How to Make Keypad Door Lock System," YouTube, Jun. 19, 2020. [Online]. Available: https://www.youtube.com/watch?v=BmzRB2z45a8.
[6] Tyzoh, "Door Lock System Using Arduino and Keypad," YouTube, Sep. 18, 2020. [Online]. Available: https://www.youtube.com/watch?v=TYzOhgUmZcw.
[7] CircuitPython Documentation, "OV7670 Camera Module with CircuitPython," Adafruit, Nov. 2024. [Online]. Available: https://docs.circuitpython.org/projects/ov7670/en/latest/.
[8] Hello Raspberry Pi, "OV7670 Camera & ST7789 IPS Display on Raspberry Pi," Hello Raspberry Pi, Aug. 2021. [Online]. Available: https://helloraspberrypi.blogspot.com/2021/08/ov7670-camera-st7789-ips-display-on.html.
[9] Arduino Forum, "Arduino Mega with LCD ST7920," Arduino Forum, Mar. 16, 2020. [Online]. Available: https://forum.arduino.cc/t/arduino-mega-with-lcd-st7920/664861/3.
[10] A. Geitgey, "Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning," Medium, Jul. 2018. [Online]. Available: https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121

[11] A. Abiyev and C. Akkaya, ”Design and Implementation of Smart Home Security System,” *ResearchGate*, Jan. 2016. [Online]. Available: https://www.researchgate.net/publication/293173717_Design_and_Implementation_of_Smart_Home_Security_System

[12] Study Tronics, ”Home Security Alarm System Using Arduino: Project Report,” *Weebly*, Jan. 2016. [Online]. Available: https://studytronics.weebly.com/uploads/4/4/3/7/44372217/home_security_alarm_system_using_arduino_project_report.pdf

[13] Google Images, ”Online Image,” [Online]. Available: https://images.app.goo.gl/HYtVVqwmN3Cmn3Wo7.

[14] Google Images, ”Online Image,” [Online]. Available: https://images.app.goo.gl/4gdvFREvFtYcHtPr7.

[15] Google Images, ”Online Image,” [Online]. Available: https://images.app.goo.gl/SEX5GFU2fdHrkyt56.

[16] Google Images, ”Online Image,” [Online]. Available: https://images.app.goo.gl/gcxykFjiUGo55qWu5.

[17] Google Images, ”Online Image,” [Online]. Available: https://images.app.goo.gl/QxWhB8UQ7yWY64V27.

[18] Google Images, ”Online Image,” [Online]. Available: https://images.app.goo.gl/qQFKrCAvczxHPGbw9.

[19] Google Images, ”Online Image,” [Online]. Available: https://images.app.goo.gl/CY3BYcAiiZTkMXPP8.

[20] Google Images, ”Online Image,” [Online]. Available: https://images.app.goo.gl/rwJdrfoQ7AE3LEpy7.

[21] Google Images, ”Online Image,” [Online]. Available: https://images.app.goo.gl/ifADJHFsstkhx4H68.

[22] Google Images, ”Online Image,” [Online]. Available: https://images.app.goo.gl/s175hPUDzqexejRi6.

[23] Google Images, ”Online Image,” [Online]. Available: https://images.app.goo.gl/LqaB4JGqUyUH4PLE8.

[24] Google Images, ”Online Image,” [Online]. Available: https://images.app.goo.gl/HM6Vpi4iiyD3jyHd7.