IoT-Based Smart Face Attendance System with Motion Detection

# Overview

This project integrates motion detection using an IR sensor, ESP32, and ThingSpeak with a smart face recognition system on a laptop. When motion is detected, a trigger is sent to ThingSpeak. The Python program reads this trigger and starts a GUI that registers or marks attendance using face recognition. After attendance is processed, the ESP32 resumes monitoring.

# Required Components

- ESP32 board  
- IR sensor  
- Buzzer  
- Jumper wires  
- Internet connectivity (Wi-Fi)  
- Laptop with webcam  
- Python packages: face\_recognition, opencv-python, pillow, requests, tkinter

# Hardware Connections

1. Connect IR sensor OUT pin to ESP32 GPIO13  
2. Connect IR sensor VCC to 3.3V and GND to GND  
3. Connect Buzzer positive to GPIO5 and negative to GND

# ESP32 Arduino Code

Use the following code in Arduino IDE to upload to your ESP32:

#include <WiFi.h>  
#include "ThingSpeak.h"  
  
const char\* ssid = "Wokwi-GUEST";   
const char\* password = "";   
  
WiFiClient client;  
unsigned long channelID = 3005560;  
const char\* writeAPIKey = "ALW22BSTI0HSVEIO";  
  
int ir = 13;  
int buzzer = 5;  
int lastState = 0;  
bool waitingForAttendance = false;  
  
void setup() {  
 pinMode(ir, INPUT);  
 pinMode(buzzer, OUTPUT);  
 Serial.begin(115200);  
  
 WiFi.begin(ssid, password);  
 while (WiFi.status() != WL\_CONNECTED) {  
 Serial.println("Connecting...");  
 delay(1000);  
 }  
  
 Serial.println("Connected to WiFi");  
 ThingSpeak.begin(client);  
}  
  
void loop() {  
 if (waitingForAttendance) {  
 // Check if attendance is done (field1 reset to 0)  
 WiFiClient client;  
 HTTPClient http;  
 http.begin(client, "https://api.thingspeak.com/channels/3005560/fields/1/last.txt");  
 int httpCode = http.GET();  
 if (httpCode == 200) {  
 String payload = http.getString();  
 if (payload.toInt() == 0) {  
 Serial.println("✅ Attendance complete, resuming monitoring...");  
 waitingForAttendance = false;  
 }  
 }  
 http.end();  
 delay(3000);  
 return;  
 }  
  
 int currentState = digitalRead(ir);  
 if (currentState == 1 && lastState == 0) {  
 Serial.println("Motion Detected!");  
 digitalWrite(buzzer, HIGH);  
  
 ThingSpeak.setField(1, 1);  
 int x = ThingSpeak.writeFields(channelID, writeAPIKey);  
 if (x == 200) {  
 Serial.println("✅ Data sent to ThingSpeak.");  
 waitingForAttendance = true;  
 } else {  
 Serial.print("❌ Failed to send. HTTP error: ");  
 Serial.println(x);  
 }  
 }  
  
 if (currentState == 0) {  
 digitalWrite(buzzer, LOW);  
 }  
  
 lastState = currentState;  
 delay(100);  
}

# Python Face Recognition Code

Ensure required Python libraries are installed and then run this script:

Due to size limitations, Python code is continued in a separate file or appendix.

# Step-by-Step Procedure

1. Connect your ESP32 with the IR sensor and buzzer as per the wiring diagram.  
2. Open Arduino IDE, paste the ESP32 code, enter your ThingSpeak channel info, and upload it to the board.  
3. Create a ThingSpeak channel with one field (field1).  
4. Install required Python libraries on your laptop:  
 pip install face\_recognition opencv-python pillow requests  
5. Save the Python script on your laptop with folders: `faces/` and `data/`  
6. Run the Python script. It will wait for ThingSpeak field1 == 1.  
7. Trigger motion in front of the IR sensor to start face recognition.  
8. Register or mark attendance from the GUI.  
9. After processing, the script resets ThingSpeak field1 to 0.  
10. ESP32 resumes monitoring for next motion.

Camera.py code to initialize :

import cv2

import face\_recognition

import os

import csv

import requests

import time

from datetime import datetime

import tkinter as tk

from PIL import Image, ImageTk

from threading import Thread

# Folder setup

os.makedirs("faces", exist\_ok=True)

os.makedirs("data", exist\_ok=True)

# ThingSpeak Configuration

THINGSPEAK\_CHANNEL\_ID = "3005560"

READ\_API\_KEY = "VHSDV3DSF21N8QFB"

WRITE\_API\_KEY = "ALW22BSTI0HSVEIO"

current\_frame = None

recognized\_name = "Unknown"

camera\_running = True

exit\_after\_success = False

def wait\_for\_trigger():

    print("🔄 Waiting for motion trigger from ThingSpeak...")

    url = f"https://api.thingspeak.com/channels/{THINGSPEAK\_CHANNEL\_ID}/fields/1/last.json?api\_key={READ\_API\_KEY}"

    while True:

        try:

            response = requests.get(url)

            if response.status\_code == 200:

                data = response.json()

                field\_value = int(data['field1']) if data['field1'] else 0

                print(f"🔍 Motion Status: {field\_value}")

                if field\_value == 1:

                    print("✅ Trigger received from ThingSpeak!")

                    return

        except Exception as e:

            print("❌ Error checking ThingSpeak:", e)

        time.sleep(3)

def reset\_thingspeak\_trigger():

    url = "https://api.thingspeak.com/update.json"

    data = {

        "api\_key": WRITE\_API\_KEY,

        "field1": 0

    }

    try:

        requests.post(url, data=data)

        print("✅ Reset ThingSpeak trigger to 0.")

    except:

        print("❌ Failed to reset trigger.")

def load\_known\_faces():

    encodings, names = [], []

    for file in os.listdir("faces"):

        if file.endswith(".jpg"):

            path = f"faces/{file}"

            img = face\_recognition.load\_image\_file(path)

            enc = face\_recognition.face\_encodings(img)

            if enc:

                encodings.append(enc[0])

                names.append(file[:-4])

    return encodings, names

def is\_face\_registered(frame):

    encodings, names = load\_known\_faces()

    rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

    faces = face\_recognition.face\_encodings(rgb)

    if not faces:

        return False, None

    for known\_encoding, known\_name in zip(encodings, names):

        match = face\_recognition.compare\_faces([known\_encoding], faces[0])[0]

        distance = face\_recognition.face\_distance([known\_encoding], faces[0])[0]

        if match and distance < 0.5:

            return True, known\_name

    return False, None

def register\_face():

    global current\_frame, exit\_after\_success

    name = name\_entry.get().strip()

    if not name:

        status\_label.config(text="⚠️ Enter name first!")

        return

    found, existing\_name = is\_face\_registered(current\_frame)

    if found:

        status\_label.config(text=f"⚠️ Face already registered as {existing\_name}")

        return

    file\_path = f"faces/{name}.jpg"

    rgb = cv2.cvtColor(current\_frame, cv2.COLOR\_BGR2RGB)

    locs = face\_recognition.face\_locations(rgb)

    if len(locs) != 1:

        status\_label.config(text="❌ Show only one face")

        return

    cv2.imwrite(file\_path, current\_frame)

    status\_label.config(text=f"✅ Registered {name}")

    exit\_after\_success = True

    root.quit()

def mark\_attendance(action):

    global recognized\_name, exit\_after\_success

    if recognized\_name == "Unknown":

        status\_label.config(text="❌ Face not recognized")

        return

    filename = f"data/{recognized\_name}.csv"

    date = datetime.now().strftime("%Y-%m-%d")

    time\_now = datetime.now().strftime("%H:%M:%S")

    if os.path.exists(filename):

        with open(filename, 'r') as f:

            lines = f.readlines()

            if any(date in line and action in line for line in lines):

                status\_label.config(text=f"⚠️ {action} already marked")

                return

            if action == "Check-out" and not any("Check-in" in line and date in line for line in lines):

                status\_label.config(text="❌ No check-in found")

                return

    with open(filename, 'a', newline='') as f:

        writer = csv.writer(f)

        writer.writerow([date, time\_now, action])

    status\_label.config(text=f"✅ {action} for {recognized\_name}")

    exit\_after\_success = True

    root.quit()

def update\_camera():

    global current\_frame, recognized\_name, camera\_running

    encodings, names = load\_known\_faces()

    cap = cv2.VideoCapture(0)

    while camera\_running:

        ret, frame = cap.read()

        current\_frame = frame.copy()

        rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

        small = cv2.resize(rgb, (0, 0), fx=0.25, fy=0.25)

        face\_locations = face\_recognition.face\_locations(small)

        face\_encodings = [face\_recognition.face\_encodings(small, [loc])[0]

                          for loc in face\_locations if face\_recognition.face\_encodings(small, [loc])]

        name = "Unknown"

        for encoding in face\_encodings:

            matches = face\_recognition.compare\_faces(encodings, encoding)

            distances = face\_recognition.face\_distance(encodings, encoding)

            if distances.any():

                best = distances.argmin()

                if matches[best] and distances[best] < 0.5:

                    name = names[best]

        recognized\_name = name

        for loc in face\_locations:

            top, right, bottom, left = [v \* 4 for v in loc]

            color = (0, 255, 0) if name != "Unknown" else (0, 0, 255)

            cv2.rectangle(frame, (left, top), (right, bottom), color, 2)

            cv2.putText(frame, name, (left, top - 10),

                        cv2.FONT\_HERSHEY\_SIMPLEX, 0.9, color, 2)

        img = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

        img = Image.fromarray(img)

        imgtk = ImageTk.PhotoImage(image=img)

        video\_label.imgtk = imgtk

        video\_label.configure(image=imgtk)

    cap.release()

# 🔁 Main Loop

while True:

    wait\_for\_trigger()

    exit\_after\_success = False

    root = tk.Tk()

    root.title("Smart Face Attendance System")

    root.geometry("850x600")

    video\_label = tk.Label(root)

    video\_label.pack()

    tk.Label(root, text="Enter Name (for Register):").pack()

    name\_entry = tk.Entry(root)

    name\_entry.pack()

    tk.Button(root, text="📸 Register", width=20, command=register\_face).pack(pady=2)

    tk.Button(root, text="✅ Check-In", width=20, command=lambda: mark\_attendance("Check-in")).pack(pady=2)

    tk.Button(root, text="🔁 Check-Out", width=20, command=lambda: mark\_attendance("Check-out")).pack(pady=2)

    tk.Button(root, text="❌ Quit", width=20, command=root.quit).pack(pady=2)

    status\_label = tk.Label(root, text="", fg="blue")

    status\_label.pack(pady=5)

    camera\_running = True

    thread = Thread(target=update\_camera)

    thread.start()

    root.protocol("WM\_DELETE\_WINDOW", root.quit)

    root.mainloop()

    camera\_running = False

    reset\_thingspeak\_trigger()

    if not exit\_after\_success:

        print("🛑 GUI manually closed. Exiting loop.")

        break

output:

