IoT-Based Smart Security System Using Face Recognition, Age, and Gender Detection

I. Introduction

Project Title: IoT-Based Smart Security System Using Face Recognition, Age, and Gender Detection

Project Overview:

The objective of this project is to develop a smart security system that utilizes face recognition to identify individuals, detect their age and gender, and alert authorities in case of an imposter. The system captures an image using a Raspberry Pi camera, processes it using OpenCV and a deep learning model, and sends an alert email if an unknown individual is detected.

Background:

With increasing security concerns in both residential and commercial spaces, automated systems using artificial intelligence and IoT can significantly enhance safety measures. Face recognition combined with age and gender classification provides an extra layer of security by ensuring that only authorized individuals are recognized and flagged.

II. Literature Review

Overview of IoT:

The Internet of Things (IoT) connects physical devices to the internet, enabling them to collect and exchange data. IoT has various applications in smart homes, healthcare, security, and industrial automation. This project leverages IoT for real-time security monitoring.

End Users of the Application:

- Homeowners seeking enhanced security
- Offices and restricted zones requiring authorized access
- Educational institutions for attendance and security
- Banking and financial institutions for access control

III. System Design and Architecture

Hardware Components:

- **Raspberry Pi** – Acts as the central processing unit.
- **PiCamera** – Captures images for processing.
- **Buzzer** – Provides an alert in case of an imposter.
- **GPIO Modules** – Controls hardware components like the buzzer.

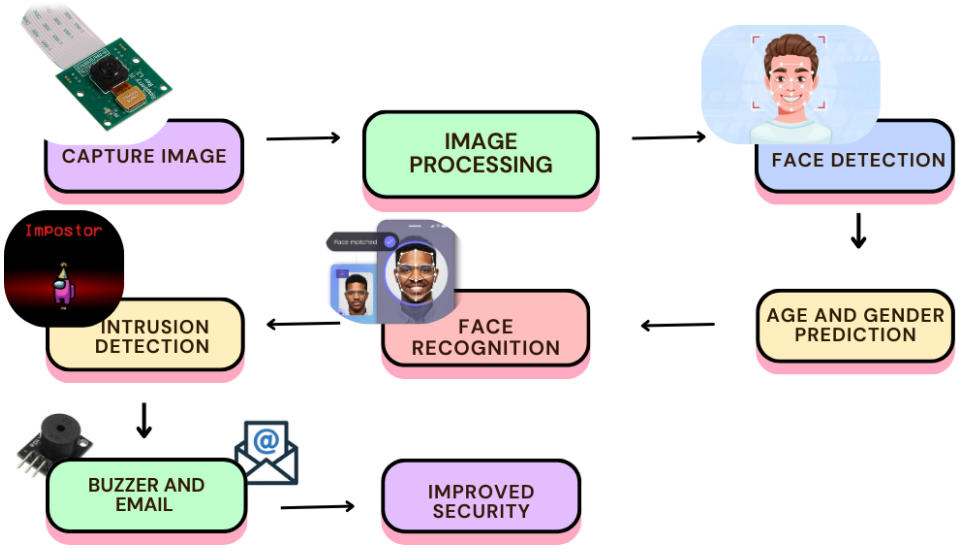
Software Components:

- **Python** – Programming language for implementation.
- **OpenCV** – For image processing and face detection.
- **Face Recognition Library** – For identifying known faces.
- **Deep Learning Models (Caffe Framework)** – For age and gender classification.
- **SMTP (Simple Mail Transfer Protocol)** – For sending email alerts.

System Architecture:

The system follows a step-by-step process:

1. The Raspberry Pi camera captures an image.
2. Face recognition is applied to detect and compare faces with known faces.
3. Age and gender classification is performed.
4. If an imposter is detected, a buzzer is activated, and an email is sent.
5. The processed image is displayed with detected labels.



IV. Implementation and Testing

Implementation Details:

1. The pre-trained models for age and gender classification were downloaded and integrated.
2. Face recognition was implemented using the **face_recognition** library.
3. The buzzer was controlled using the Raspberry Pi GPIO module.
4. The SMTP library was used to send email alerts with an attached image.

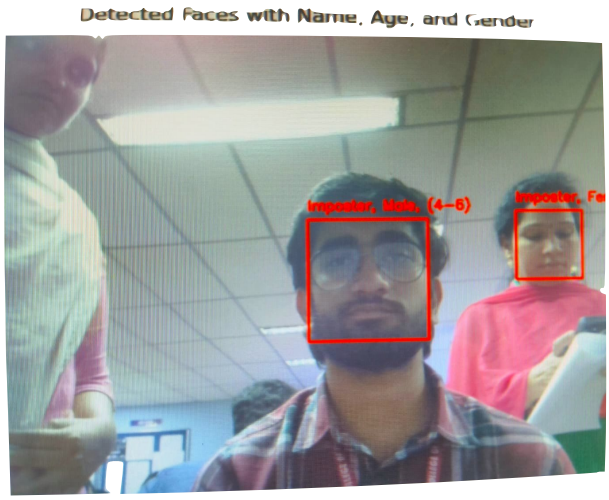
Testing and Validation:

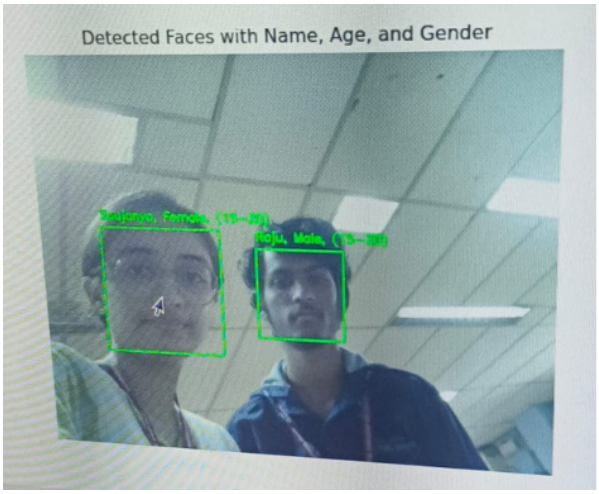
- **Face Recognition Testing:** The system was tested with a dataset of known individuals.
- **Age and Gender Detection Accuracy:** The system was evaluated for correct classifications.
- **Buzzer and Email Alert Testing:** The buzzer and email functionalities were validated for imposter detection.

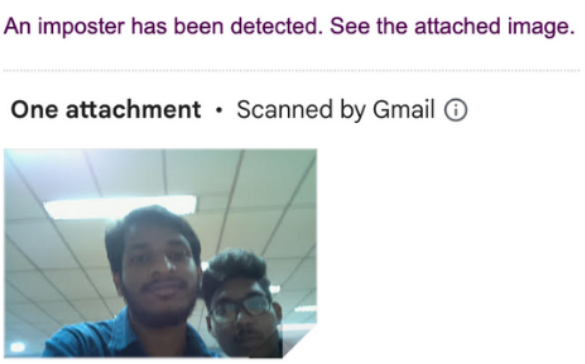
Known faces :



Imposter testing:







V. Results and Discussion

Results:

- Successfully recognized known faces with high accuracy.
- Imposters were correctly identified and alerted through a buzzer and email.
- Age and gender classification provided reasonable accuracy based on test cases.

Discussion:

- The face recognition accuracy depended on lighting conditions and image resolution.
- The email alert feature ensured real-time security monitoring.
- Future improvements could include a database to log visitor activity.

VI. Conclusion and Future Work

Conclusion:

This project demonstrated an IoT-based security system capable of detecting, classifying, and alerting authorities in case of an imposter. The combination of AI, IoT, and deep learning enhanced the overall security infrastructure.

Future Work:

- **Enhancing Face Recognition**: Improving recognition accuracy with more training data.
- **Cloud Integration**: Storing captured images and logs in a cloud database.
- **Mobile App Integration**: Providing real-time notifications to users via a mobile application.
- **Additional Biometrics**: Adding voice or fingerprint recognition for multi-factor authentication.

VII. References

- OpenCV documentation: https://opencv.org
- Face Recognition library: https://github.com/ageitgey/face_recognition
- Raspberry Pi official site: https://www.raspberrypi.org
- Caffe model for age and gender prediction : https://github.com/tringn/AgeGenderPrediction
- Dlib Face Recognition Library – http://dlib.net/
- Email Automation with Python (smtplib) – https://docs.python.org/3/library/smtplib.html
- **Raspberry Pi GPIO Library Documentation** – https://gpiozero.readthedocs.io/en/stable/
- **Face Detection with Deep Learning (MTCNN)** – https://github.com/ipazc/mtcnn

VIII. Appendices

Additional Materials:

- Source code implementation :

```
import cv2
import numpy as np
import face_recognition
from picamera import PiCamera
import time
import os
import smtplib
import ssl
from email.message import EmailMessage
import RPi.GPIO as GPIO
import matplotlib.pyplot as plt

# --- CONFIGURATION ---

# Paths to pre-trained models
age_proto = "/home/iot/opencv_models/deploy_age.prototxt"
age_model = "/home/iot/Downloads/age_net.caffemodel"
gender_proto = "/home/iot/opencv_models/deploy_gender.prototxt"
gender_model = "/home/iot/Downloads/gender_net.caffemodel"

# Email credentials (use app password for Gmail)
sender_email = "ramasoujanya9@gmail.com"
receiver_email = "322103310193@gvpce.ac.in"  # Replace with the actual email
email_password = "ynzj uhmg kivo zlrr"

# Buzzer GPIO setup
buzzer_pin = 18  # Change based on your wiring
GPIO.setmode(GPIO.BCM)
GPIO.setup(buzzer_pin, GPIO.OUT)

# Load the models
age_net = cv2.dnn.readNet(age_model, age_proto)
gender_net = cv2.dnn.readNet(gender_model, gender_proto)

# Check if models loaded correctly
if age_net.empty() or gender_net.empty():
    print("Error: Could not load age or gender model. Check file paths.")
    exit()

# Define age and gender categories
age_labels = ['(0-2)', '(4-6)', '(8-12)', '(15-20)', '(25-32)', '(38-43)', '(48-53)', '(60-100)']
gender_labels = ['Male', 'Female']

# Load known faces
```

```
known_face_encodings = []
known_face_names = []
known_faces_dir = "known_faces"

for filename in os.listdir(known_faces_dir):
    if filename.endswith(".jpg") or filename.endswith(".png"):
        image_path = os.path.join(known_faces_dir, filename)
        image = face_recognition.load_image_file(image_path)
        encoding = face_recognition.face_encodings(image)
        if encoding:
            known_face_encodings.append(encoding[0])
            known_face_names.append(os.path.splitext(filename)[0])  # Name from filename

# --- FUNCTIONS ---

def beep_buzzer(duration=1):
    """Activates the buzzer for a specified duration before capturing an image."""
    GPIO.output(buzzer_pin, GPIO.HIGH)
    time.sleep(duration)
    GPIO.output(buzzer_pin, GPIO.LOW)

def send_email(image_path):
    """Sends an email with the captured imposter image."""
    msg = EmailMessage()
    msg["Subject"] = "Security Alert: Imposter Detected"
    msg["From"] = sender_email
    msg["To"] = receiver_email
    msg.set_content("An imposter has been detected. See the attached image.")

    with open(image_path, "rb") as img:
                            msg.add_attachment(img.read(),  maintype="image",  subtype="jpeg",
filename="imposter.jpg")

    context = ssl.create_default_context()
    with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context) as server:
        server.login(sender_email, email_password)
        server.send_message(msg)
        print("Email sent successfully!")

# --- IMAGE CAPTURE & PROCESSING ---

# Alert before capturing
print("Buzzer alert before capturing the image...")
beep_buzzer(1)  # Beep for 1 second

# Capture image using PiCamera
image_path = "captured_image.jpg"
camera = PiCamera()
camera.resolution = (640, 480)
time.sleep(2)  # Allow camera to warm up
camera.capture(image_path)
```

```
camera.close()
print("Picture captured successfully! Processing...")

# Read the captured image
image = cv2.imread(image_path)
rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Detect faces and extract encodings
face_locations = face_recognition.face_locations(rgb_image)
face_encodings = face_recognition.face_encodings(rgb_image, face_locations)
print(f"Number of faces detected: {len(face_locations)}")

# Store known person's name
recognized_person = None  # Track the first recognized person

# Store names for all detected faces
face_names = []

# Process each detected face
for (top, right, bottom, left), face_encoding in zip(face_locations, face_encodings):
    # Compare detected face with known faces
    name = "Imposter"
    face_distances = face_recognition.face_distance(known_face_encodings, face_encoding)

    if len(face_distances) > 0:
        best_match_index = np.argmin(face_distances)  # Find the closest match
        if face_distances[best_match_index] < 0.5:  # Adjust threshold for stricter matching
            name = known_face_names[best_match_index]
            if recognized_person is None:
                recognized_person = name  # Store the first recognized person

    face_names.append(name)  # Store names before processing age and gender

# Update unknown faces' names if a known person is detected
# Correct logic: Only assign "friend" to true imposters
for i in range(len(face_names)):
    if face_names[i] == "Imposter":
        # Check if there is at least one known person detected
        if recognized_person:
            face_names[i] = f"{recognized_person}'s friend"
        else:
            face_names[i] = "Imposter"  # Remain as an imposter


# Process and display detected faces with updated names
for ((top, right, bottom, left), name) in zip(face_locations, face_names):
    # Extract face region for age & gender detection
    face = image[top:bottom, left:right]
    blob = cv2.dnn.blobFromImage(face, scalefactor=1.0, size=(227, 227),
                                 mean=(78.4263377603, 87.7689143744, 114.895847746),
                                 swapRB=False, crop=False)
```

```
    # Predict Gender
    gender_net.setInput(blob)
    gender_pred = gender_net.forward()
    gender = gender_labels[gender_pred[0].argmax()]

    # Predict Age
    age_net.setInput(blob)
    age_pred = age_net.forward()
    age = age_labels[age_pred[0].argmax()]

    # Set color based on gender and imposter status
    if name == "Imposter":
        box_color = (0, 0, 255)  # Red
    elif gender == "Male":
        box_color = (255, 0, 0)  # Blue
    else:
        box_color = (255, 105, 180)  # Pink

    # Draw rectangle around face
    cv2.rectangle(image, (left, top), (right, bottom), box_color, 2)

    # Display text (name, age, gender)
    text = f"{name}, {gender}, {age}"
    cv2.putText(image, text, (left, top - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, box_color, 2)

    # Trigger buzzer and email if an imposter is detected
    if name == "Imposter":
        beep_buzzer(2)  # Beep for 2 seconds if imposter detected
        send_email(image_path)

# Display the final image
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.title("Detected Faces with Name, Age, and Gender")
plt.show()

# Cleanup GPIO
GPIO.cleanup()
```