SAFEHOMECAM

INTRODUCTION

"SafeHomeCam" is an AI-based home security system designed to recognize faces and hand gestures to trigger specific safety actions.
It combines computer vision, deep learning, and IoT alerting mechanisms to ensure home safety and real-time responsiveness.

The system uses:
• Hand gesture recognition (via a trained CNN model)
• Face recognition for user authentication
• Edge detection and image enhancement for clear input
• Audio alerts and Twilio integration (for message/call alerts)

This project demonstrates how artificial intelligence can be integrated into home automation systems for real-time surveillance, access control, and alert generation.

HYPOTHESIS

By integrating gesture and face recognition modules with edge detection and safety automation, it is hypothesized that:

• The system can accurately recognize hand gestures (Help, Call, Danger, Thumbs Up/Down).
• Unauthorized users will be detected and trigger alarms during SafeHouse Mode.
• Edge-enhancement and contrast techniques will improve detection accuracy in varied lighting conditions.
• Overall, "SafeHomeCam" will enhance home safety through a reliable and intelligent camera-based monitoring system.

OBJECTIVES

• To implement real-time gesture and face recognition using OpenCV and cvzone.
• To enhance camera input using Gamma correction and edge detection.
• To activate audio and alert mechanisms on recognizing danger gestures or unknown faces.
• To design a scalable, easy-to-use safety system for smart homes.

PROCEDURE

1. Setup Environment
  - Install dependencies: opencv-python, cvzone, numpy, playsound, face_recognition, twilio, threading.
  - Initialize webcam and detectors.

2. Face Recognition
  - Capture and encode known user faces.
  - Compare live faces with the stored encodings.
  - If face is “Unknown” during SafeHouse Mode, trigger siren.

3. Hand Gesture Detection
  - Detect hand using cvzone’s HandDetector.
  - Preprocess image using:
    • Grayscale conversion
    • Power law (Gamma) transformation
    • Contrast stretching
    • Edge detection (Canny)
  - Classify gesture using pre-trained Keras model.

4. SafeHouse Mode
  - Activated by Thumbs Up gesture.
  - Deactivated by Thumbs Down (only if recognized face is owner).
  - Alerts triggered for Help, Call, and Danger gestures.

5. Alert Mechanism
  - Play alert sounds (alarm.mp3, siren.mp3, Danger.wav).
  - (Optional) Send SMS/call via Twilio API.

CODE IMPLEMENTATION

Main.py :-

```
import cv2
from cvzone.HandTrackingModule import HandDetector
from cvzone.ClassificationModule import Classifier
import numpy as np
import math
import time
from playsound import playsound
from twilio.rest import Client
import face_recognition
import os
import threading
import csv
from datetime import datetime

# ========================== CSV LOGGING ==========================
log_file = os.path.join(os.getcwd(), "gesture_log.csv")

# Create CSV file with headers if not present
if not os.path.exists(log_file):
    with open(log_file, mode='w', newline='', encoding='utf-8') as f:
        writer = csv.writer(f)
        writer.writerow(["Timestamp", "Gesture", "FaceName", "SafeHouseMode", "Message"])

# ========================== CAMERA & DETECTORS
==========================
cap = cv2.VideoCapture(0)
```

```
hand_detector = HandDetector(maxHands=1)

# ========================== FACE RECOGNITION ==========================
faces_path = r"C:\Users\prath\OneDrive\Desktop\SafeHomeCam\Data\faces"
images = []
classNames = []
if not os.path.exists(faces_path):
    os.makedirs(faces_path)

# Load faces from subfolders (Ayush, Mheet, etc.)
for person_name in os.listdir(faces_path):
    person_folder = os.path.join(faces_path, person_name)
    if not os.path.isdir(person_folder):
        continue
    for img_file in os.listdir(person_folder):
        img_path = os.path.join(person_folder, img_file)
        curImg = cv2.imread(img_path)
        if curImg is None:
            continue
        images.append(curImg)
        classNames.append(person_name)

def findEncodings(images):
    encodeList = []
    for img in images:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
        encodings = face_recognition.face_encodings(img)
        if encodings:
            encodeList.append(encodings[0])
    return encodeList

encodeListKnown = findEncodings(images)

# ========================== GESTURE CLASSIFIER ==========================
classifier = Classifier(
    r"C:\Users\prath\OneDrive\Desktop\SafeHomeCam\Model\keras_model.h5",
    r"C:\Users\prath\OneDrive\Desktop\SafeHomeCam\Model\labels.txt"
)

with open(r"C:\Users\prath\OneDrive\Desktop\SafeHomeCam\Model\labels.txt", "r") as f:
    labels = [line.strip() for line in f.readlines()]

labels = [l.split(maxsplit=1)[-1] if len(l.split()) > 1 else l for l in labels]

offset = 20
imgSize = 300

# ========================== TWILIO SETUP ==========================
account_sid = "ACa8c3a6ec4e9809e86bd009471a4a4473"
auth_token = "5395c627b65140ca30ef8e682f4c79ce"
client = Client(account_sid, auth_token)
```

```
twilio_number = "+12298059944"

owner_number = "+918623083659"

police_number = "+918623083659"


# ========================== ALERT & SAFEHOUSE SETTINGS
==========================

last_label = None

last_face_label = None

last_face_location = None

gesture_start_time = 0

triggered = False

hold_duration = 3

frame_count = 0

process_every_n_frames = 3

status_text = ""

status_expire = 0

cooldown_seconds = 5

last_trigger_time = {"Help":0, "Call":0, "Danger":0, "ThumbsUp":0, "ThumbsDown":0}


alarm_path = os.path.join(os.getcwd(), "alarm.wav")

danger_path = os.path.join(os.getcwd(), "Danger.wav")

siren_path = os.path.join(os.getcwd(), "siren.mp3")


safehouse_mode = False

unknown_start_time = 0

unknown_hold_duration = 3
```

```
# ========================== UTILITY FUNCTIONS ==========================
def log_event(gesture, face_name, message=""):
    """Log gesture events with face recognition info into a CSV."""
    try:
        with open(log_file, mode='a', newline='', encoding='utf-8') as f:
            writer = csv.writer(f)
            writer.writerow([
                datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
                gesture,
                face_name if face_name else "Unknown",
                "ON" if safehouse_mode else "OFF",
                message
            ])
    except Exception as e:
        print(f"[LOG ERROR] Could not write to CSV: {e}")

def safe_play(path):
    if os.path.isfile(path):
        try:
            from playsound import playsound
            playsound(path)
        except:
            pass

def set_status(text, duration=3):
```

```
    global status_text, status_expire
    status_text = text
    status_expire = time.time() + duration

def send_sms_sync(to, message):
    try:
        msg = client.messages.create(body=message, from_=twilio_number, to=to)
        return ("ok", getattr(msg, "sid", None))
    except Exception as e:
        return ("error", str(e))

def make_call_sync(to, message):
    try:
        call = client.calls.create(twiml=f'<Response><Say>{message}</Say></Response>',
from_=twilio_number, to=to)
        return ("ok", getattr(call, "sid", None))
    except Exception as e:
        return ("error", str(e))

def async_send_sms(to, message):
    def job():
        res = send_sms_sync(to, message)
        set_status(f"SMS -> {to}: {res[0]}", 4)
    threading.Thread(target=job, daemon=True).start()

def async_make_call(to, message):
```

```
    def job():
        res = make_call_sync(to, message)
        set_status(f"Call -> {to}: {res[0]}", 4)
    threading.Thread(target=job, daemon=True).start()

def trigger_actions(label):
    global safehouse_mode, unknown_start_time, last_face_label

    gesture_name = label.replace(" ", "").strip()

    if gesture_name not in ["Help", "Call", "Danger", "ThumbsUp", "ThumbsDown"]:
        return

    now = time.time()
    if now - last_trigger_time.get(gesture_name, 0) < cooldown_seconds:
        set_status(f"{gesture_name} (cooldown)", 2)
        return
    last_trigger_time[gesture_name] = now

    # ============================ SAFEHOUSE CONTROL (Only Mheet)
============================
    if gesture_name == "ThumbsUp":
        if last_face_label == "Pratham":
            safehouse_mode = True
            unknown_start_time = 0
            set_status("SafeHouse Mode ON (Authorized: Mheet)", 5)
```

```
            log_event("ThumbsUp", last_face_label, "SafeHouse Mode turned ON")
        else:
            set_status("Access Denied: Only Mheet can turn ON SafeHouse Mode", 5)
            log_event("ThumbsUp", last_face_label, "Access Denied")

    elif gesture_name == "ThumbsDown":
        if last_face_label == "Mheet":
            safehouse_mode = False
            set_status("SafeHouse Mode OFF (Authorized: Mheet)", 5)
            unknown_start_time = 0
            log_event("ThumbsDown", last_face_label, "SafeHouse Mode turned OFF")
        else:
            set_status("Access Denied: Only Mheet can turn OFF SafeHouse Mode", 5)
            log_event("ThumbsDown", last_face_label, "Access Denied")

    elif gesture_name == "Help":
        set_status("HELP triggered", 5)
        threading.Thread(target=safe_play, args=(alarm_path,), daemon=True).start()
        async_send_sms(owner_number, "🆘 HELP detected! Immediate assistance may be
required.")
        log_event("Help", last_face_label, "HELP gesture triggered")

    elif gesture_name == "Call":
        set_status("CALL triggered", 5)
        async_make_call(owner_number, "Emergency call request received. Please check
immediately.")
        async_send_sms(owner_number, "📞 CALL gesture detected. Call initiated.")
```

```
        log_event("Call", last_face_label, "CALL gesture triggered")


    elif gesture_name == "Danger":
        set_status("DANGER triggered", 6)
        threading.Thread(target=safe_play, args=(danger_path,), daemon=True).start()
        async_send_sms(owner_number, "⚠ DANGER ALERT! Something unusual detected.")
        async_send_sms(police_number, "🚑 Possible threat detected at the registered address.")
        async_make_call(owner_number, "Danger alert triggered! Authorities have been notified.")
        log_event("Danger", last_face_label, "DANGER gesture triggered")


# ========================= MAIN LOOP =========================
while True:
    success, img = cap.read()
    if not success:
        break
    imgOutput = img.copy()
    frame_h, frame_w = img.shape[:2]


    hands, img = hand_detector.findHands(img, flipType=False)
    detected_region = None
    region_type = None
    label = None
    frame_count += 1


    # ----- PRIORITIZE HAND if visible -----
```

```
    if hands:
        hand = hands[0]
        x, y, w, h = hand['bbox']
        detected_region = (x, y, w, h)
        region_type = "hand"

# ----- Run face recognition every 5th frame (always, even with hands)
    frame_count += 1
    if frame_count % 5 == 0:
        imgS = cv2.resize(img, (0, 0), None, 0.25, 0.25)
        imgS = cv2.cvtColor(imgS, cv2.COLOR_BGR2RGB)
        facesCurFrame = face_recognition.face_locations(imgS)
        encodesCurFrame = face_recognition.face_encodings(imgS, facesCurFrame)

        if facesCurFrame:
            for encodeFace, faceLoc in zip(encodesCurFrame, facesCurFrame):
                matches = face_recognition.compare_faces(encodeListKnown, encodeFace)
                faceDis = face_recognition.face_distance(encodeListKnown, encodeFace)
                matchIndex = np.argmin(faceDis)

                if matches[matchIndex]:
                    name = classNames[matchIndex]
                    last_face_label = name
                    last_face_location = faceLoc
                else:
                    last_face_label = "Unknown"
```

```
                last_face_location = faceLoc
        else:
            last_face_label = None
            last_face_location = None

    # ---- Draw last known face box every frame ----
    if last_face_location is not None:
        y1, x2, y2, x1 = last_face_location
        y1, x2, y2, x1 = y1 * 4, x2 * 4, y2 * 4, x1 * 4
        cv2.rectangle(imgOutput, (x1, y1), (x2, y2), (255, 0, 255), 2)
        cv2.putText(imgOutput, last_face_label, (x1, y2 + 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.75, (255, 0, 255), 2)

    # ================= SAFEHOUSE: unknown person hold detection (one-time alert +
limited capture) =================

    if safehouse_mode:
        if last_face_label == "Unknown":
            if unknown_start_time == 0:
                unknown_start_time = time.time()
            elif time.time() - unknown_start_time >= unknown_hold_duration:
                if not captured_once:
                    set_status("UNKNOWN detected during SafeHouse Mode!", 6)
                    threading.Thread(target=safe_play, args=(siren_path,), daemon=True).start()

                    # ---- Create folder if not exists ----
```

```
            capture_dir = os.path.join(os.getcwd(), "Captured_Frames")
            os.makedirs(capture_dir, exist_ok=True)

            # ---- Capture up to 5 images ----
            timestamp = time.strftime("%Y%m%d_%H%M%S")
            for i in range(5):
                filename = os.path.join(capture_dir, f"Unknown_{timestamp}_{i+1}.jpg")
                cv2.imwrite(filename, imgOutput)
                time.sleep(0.3)

            # ---- Send alerts only once ----
            async_send_sms(owner_number, "Unknown person detected during SafeHouse Mode! 5 images captured.")
            async_send_sms(police_number, "Possible intrusion detected at SafeHouse.")
            async_make_call(owner_number, "Unknown person detected during SafeHouse Mode. Authorities have been notified.")

            captured_once = True  # prevent repeating siren & alerts
            log_event("UnknownFace", "Unknown", "Unknown person detected during SafeHouse Mode")
        else:
            # Reset when a known person appears
            unknown_start_time = 0
            captured_once = False

    # ================ HAND PROCESSING & CLASSIFICATION ================
    if detected_region and region_type == "hand":
```

```
x, y, w, h = detected_region
imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255
y1, y2 = max(0, y - offset), min(frame_h, y + h + offset)
x1, x2 = max(0, x - offset), min(frame_w, x + w + offset)
imgCrop = img[y1:y2, x1:x2]

if imgCrop.size != 0:
   # ---- Exp 4–5 : Grayscale Conversion (Filtering / Sharpening) ----
   gray = cv2.cvtColor(imgCrop, cv2.COLOR_BGR2GRAY)
    # ---- Exp 2 : Contrast Stretching ----
   min_val, max_val = np.min(gray), np.max(gray)
   if max_val - min_val > 0:
      gray = ((gray - min_val) / (max_val - min_val)) * 255
   gray = np.uint8(gray)
   # ---- Exp 1 : Power Law Transformation (Gamma Correction) ----
   gamma = 1.5
   gray = np.array(255 * (gray / 255) ** gamma, dtype='uint8')
    # ---- Exp 8 : Edge Detection (Canny) ----
   edges = cv2.Canny(gray, 100, 200)
   # ---- Convert Back for Display / Classification ----
   imgEnhanced = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)

   aspectRatio = h / w
   try:
      if aspectRatio > 1:
         k = imgSize / h
```

```
            wCal = math.ceil(k * w)
            imgResize = cv2.resize(imgCrop, (wCal, imgSize))
            wGap = math.ceil((imgSize - wCal) / 2)
            imgWhite[:, wGap:wCal + wGap] = imgResize
        else:
            k = imgSize / w
            hCal = math.ceil(k * h)
            imgResize = cv2.resize(imgCrop, (imgSize, hCal))
            hGap = math.ceil((imgSize - hCal) / 2)
            imgWhite[hGap:hCal + hGap, :] = imgResize
    except Exception:
        pass

    cv2.imshow('ImageCrop_Enhanced', imgEnhanced)
    cv2.imshow('ImageWhite', imgWhite)

    try:
        prediction, index = classifier.getPrediction(imgWhite, draw=False)
        if index < len(labels):
            label = labels[index]
    except Exception:
        pass

# ================= DRAWING =================
if detected_region:
    x, y, w, h = detected_region
```

```
        color = (0, 255, 0) if region_type == "hand" else (255, 0, 0)
        cv2.rectangle(imgOutput, (x - offset, y - offset), (x + w + offset, y + h + offset), color, 3)
        label_text = label if label is not None else ""
        if region_type == "face":
            cv2.putText(imgOutput, f"FACE: {label_text}", (x, y - 15),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, color, 2)
        else:
            cv2.putText(imgOutput, f"HAND: {label_text}", (x, y - 15),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, color, 2)


    if label:
        cv2.putText(imgOutput, f"Detected: {label}", (10, 50), cv2.FONT_HERSHEY_SIMPLEX,
1.0, (0,0,255), 2)


    if label and region_type == "hand":
        current_time = time.time()

        # Reset timer if a new gesture appears
        if label != last_label:
            last_label = label
            gesture_start_time = current_time
            triggered = False

        elapsed = current_time - gesture_start_time

        # Only if the same gesture is held continuously
        if elapsed < hold_duration:
```

```
            pct = elapsed / hold_duration
            cv2.rectangle(imgOutput, (10, 80), (int(10 + 200 * pct), 100), (0, 255, 0), -1)
            cv2.rectangle(imgOutput, (10, 80), (210, 100), (255, 255, 255), 2)
            remaining = max(0, int(hold_duration - elapsed))
            cv2.putText(imgOutput, f"Holding {remaining}s", (220, 95),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1)
        elif not triggered:
            triggered = True
            set_status(f"{label} gesture triggered after 3s", 4)
            trigger_actions(label)
    else:
        # Reset timer if hand not visible
        last_label = None
        gesture_start_time = 0
        triggered = False


    if safehouse_mode:
        cv2.putText(imgOutput, "SAFEHOUSE MODE ON", (10, 130),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,255,0), 2)
        if last_face_label == "Unknown" and unknown_start_time != 0:
            remaining = max(0, int(unknown_hold_duration - (time.time() -
unknown_start_time)))
            cv2.putText(imgOutput, f"Unknown hold: {remaining}s", (10, 160),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,200,200), 2)


    if status_text and time.time() < status_expire:
```

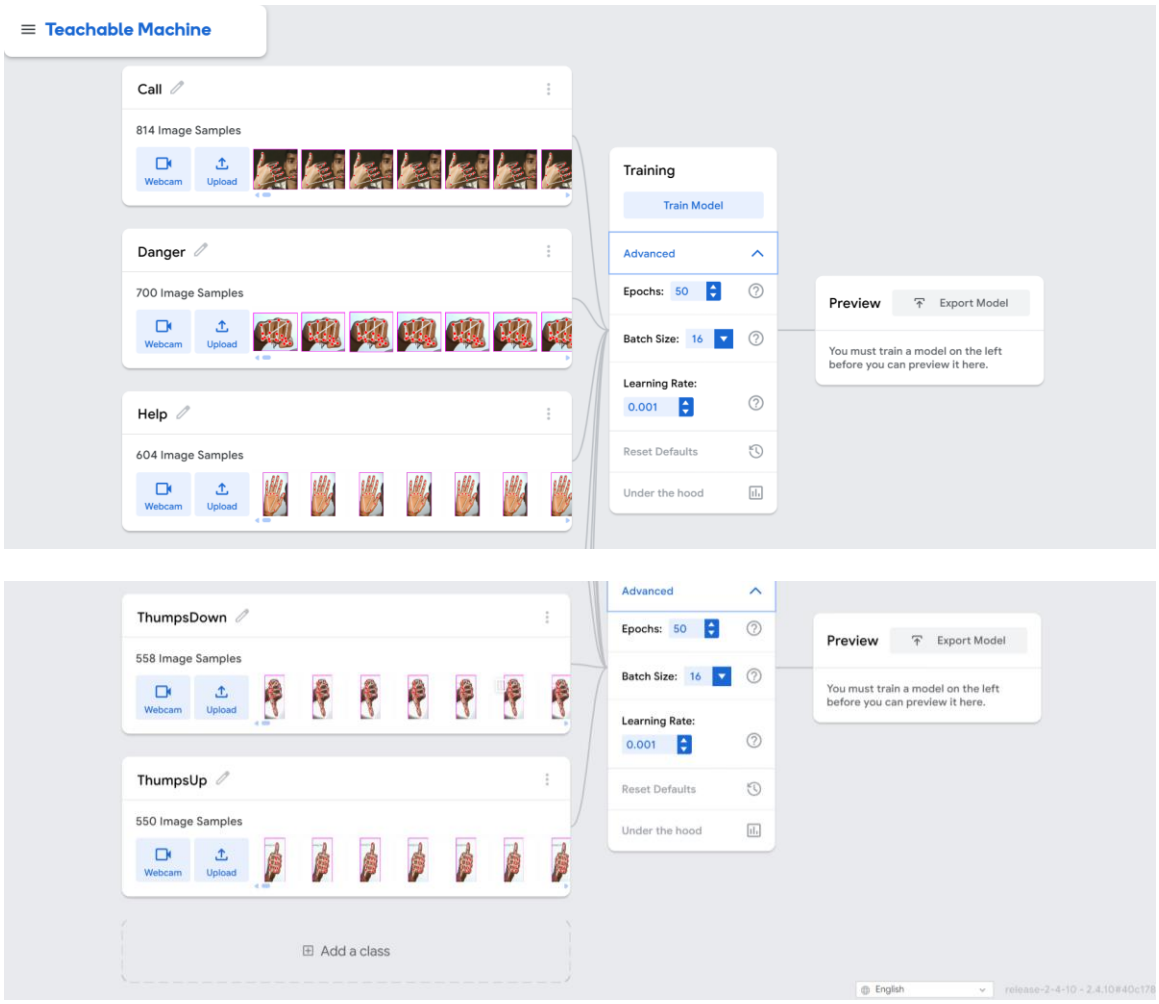
```
        cv2.putText(imgOutput, status_text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.8,
(0,0,255), 2)
    elif status_text and time.time() >= status_expire:
        status_text = ""

    cv2.namedWindow("SafeHomeCam", cv2.WINDOW_NORMAL)
    cv2.setWindowProperty("SafeHomeCam", cv2.WND_PROP_TOPMOST, 1)
    cv2.imshow("SafeHomeCam", imgOutput)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

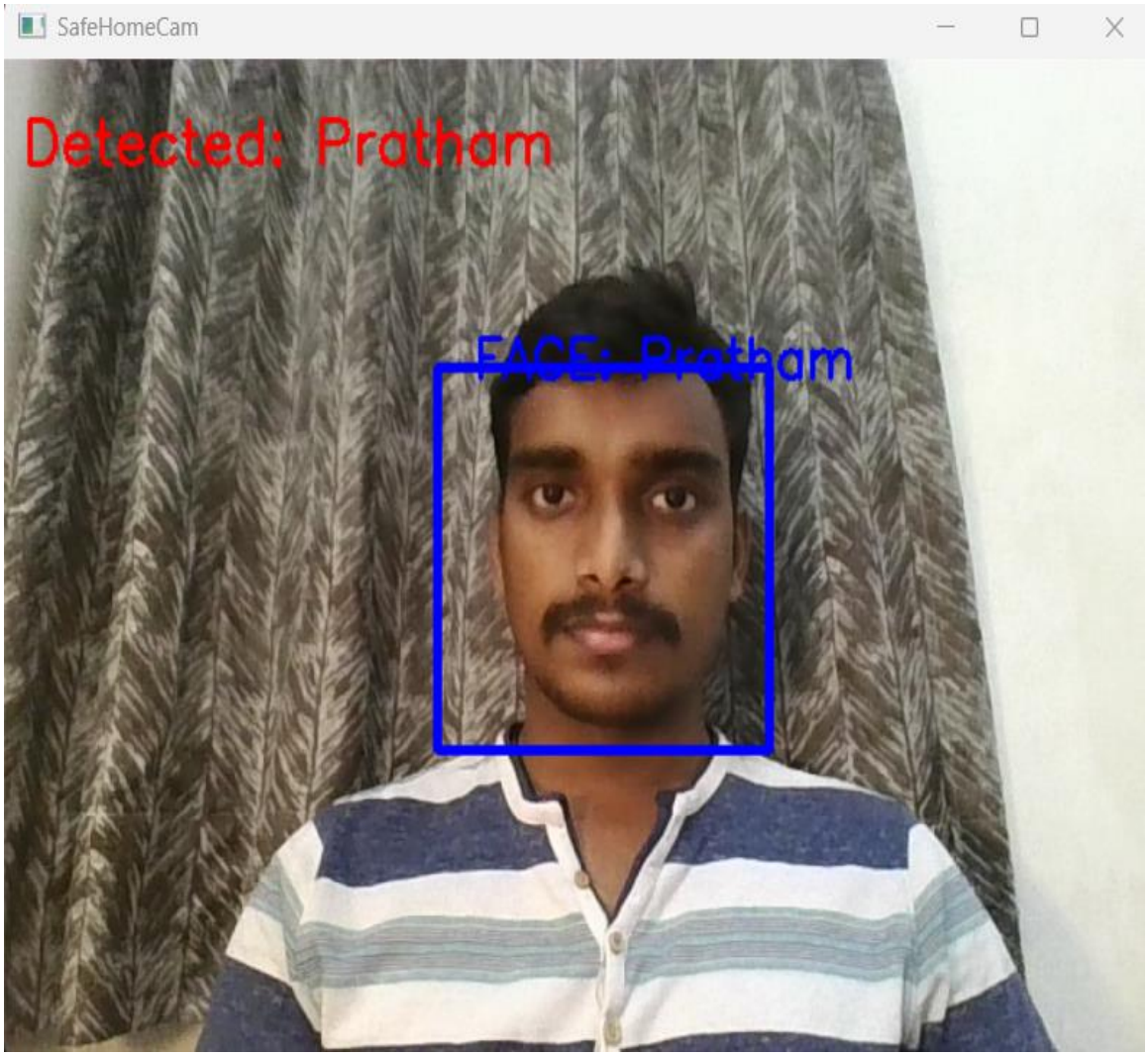
Training and Testing of the model using Teachable Machine

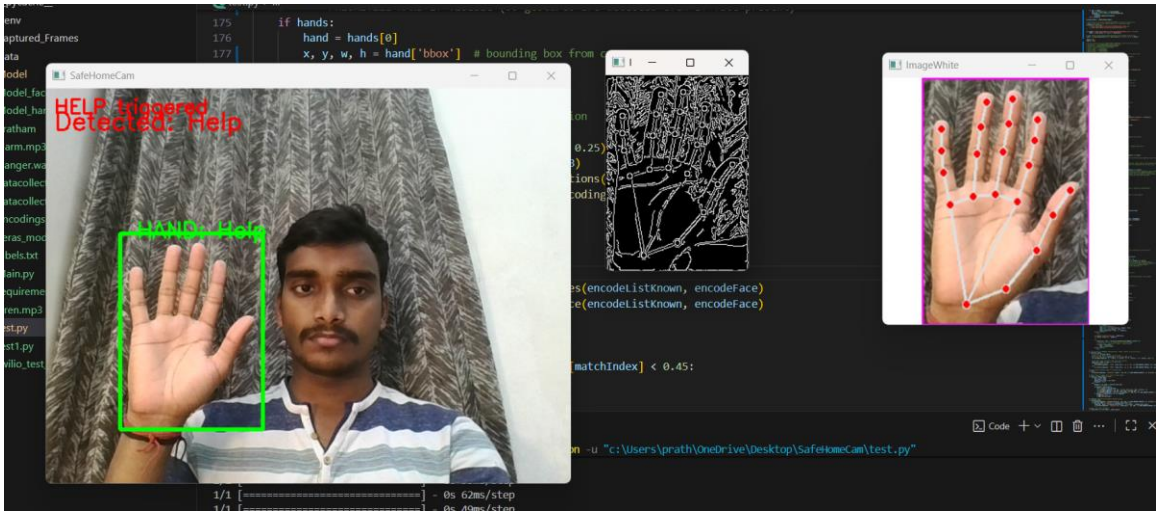


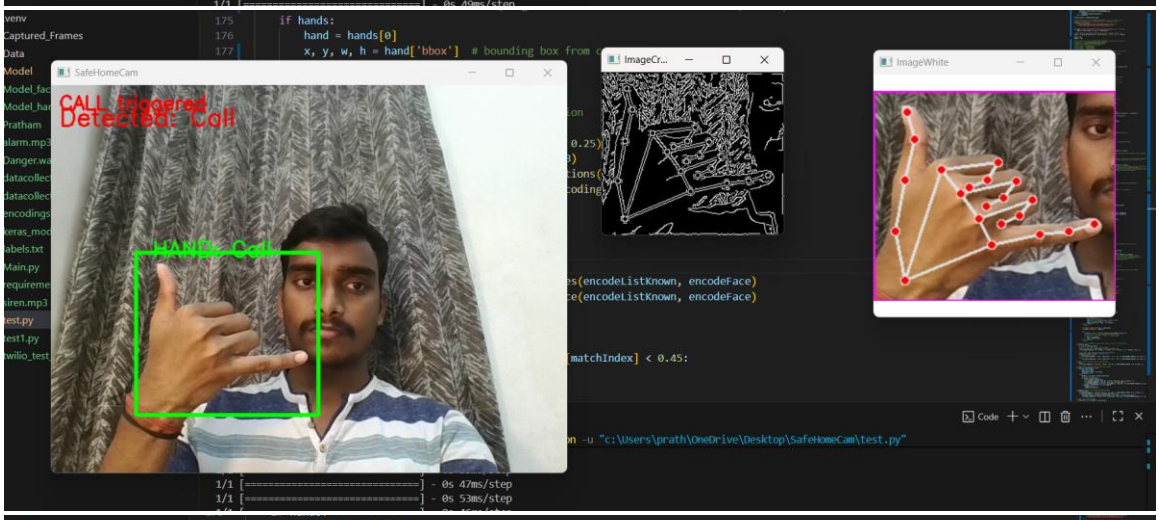
RESULTS

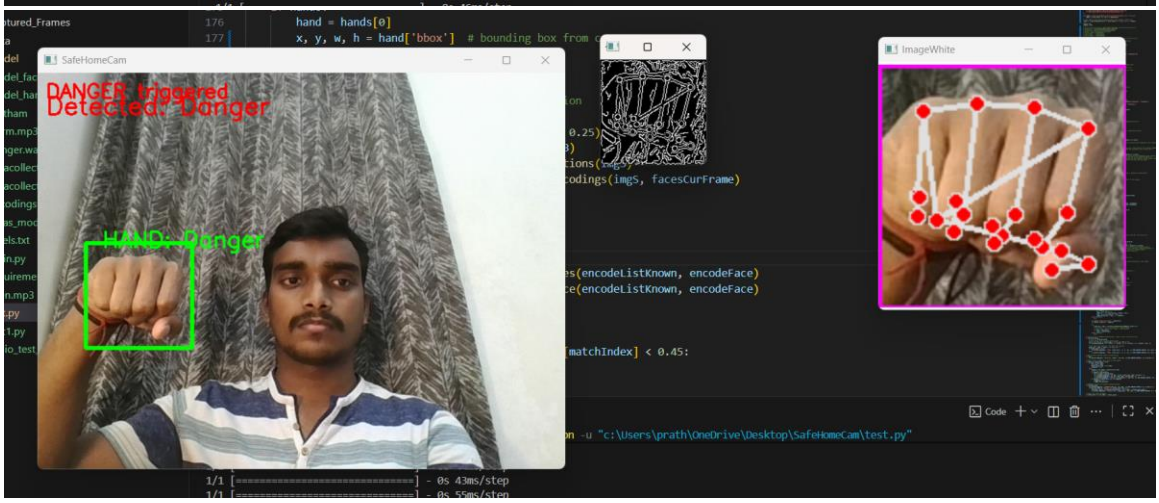
Test Case	Input	System Response
Known face + Thumbs Up	Gesture detected	SafeHouse Mode ON
Unknown face + SafeHouse Mode	Detected for 3 seconds	Siren and alert message
Help gesture	Detected	Alarm sound triggered
Thumbs Down (with owner face)	Detected	SafeHouse Mode OFF

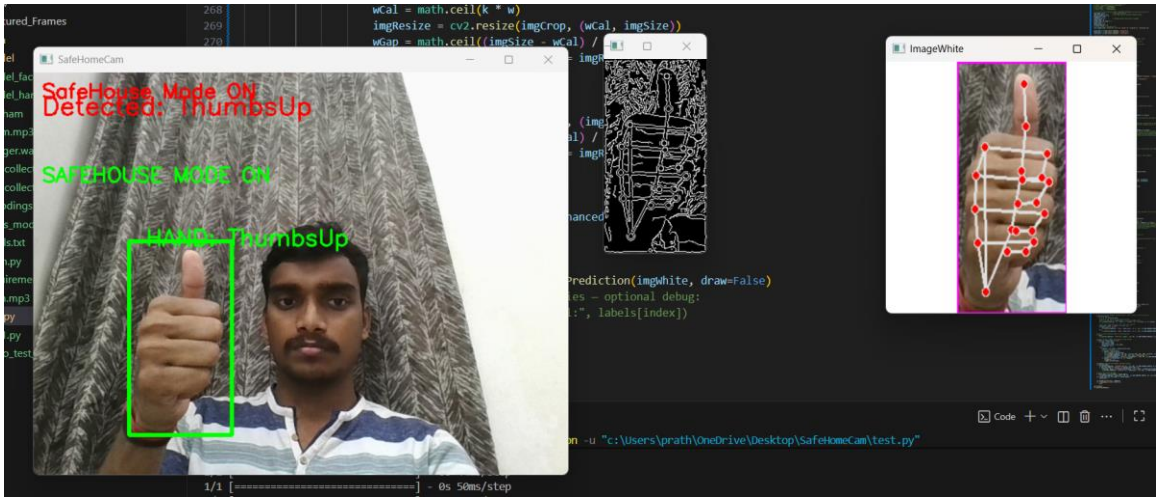
Here is the live camera feed output :-

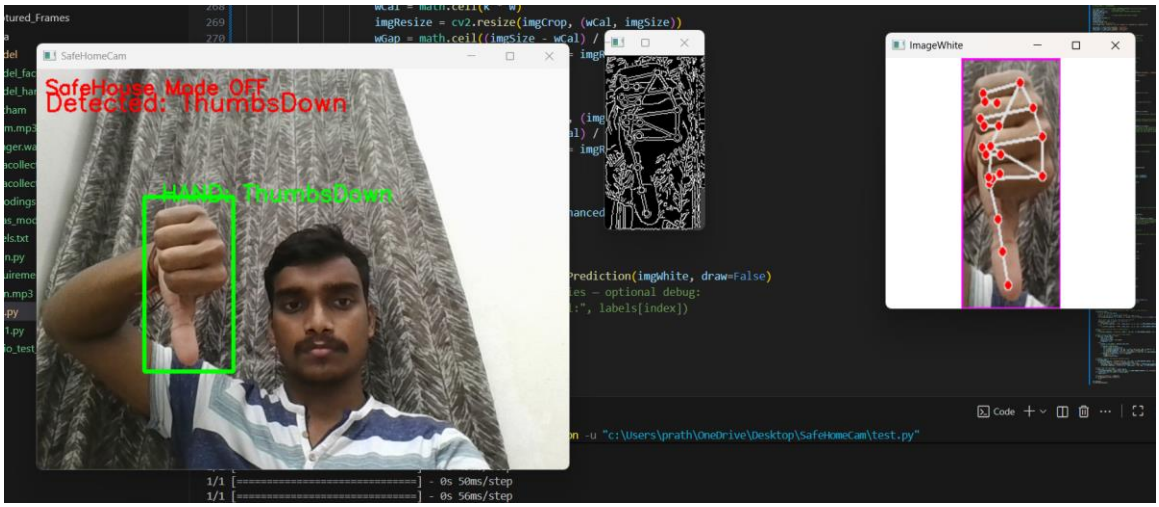

SafeHomeCam
Detected: Pratham
FACE: Pratham


SafeHomeCam
HELP triggered
Detected: Help
HAND: Help
ImageWhite
if hands:
hand = hands[0]
x, y, w, h = hand['bbox'] # bounding box from
matchIndex] < 0.45:
1/1 [==========] - 0s 62ms/step


SafeHomeCam
CALL triggered
Detected: Call
HAND: Call
ImageCr...
ImageWhite
if hands:
hand = hands[0]
x, y, w, h = hand['bbox'] # bounding box from
matchIndex] < 0.45:
1/1 [==========] - 0s 47ms/step
1/1 [==========] - 0s 53ms/step


SafeHomeCam
DANGER triggered
Detected: Danger
HAND: Danger
ImageWhite
hand = hands[0]
x, y, w, h = hand['bbox'] # bounding box from
codings(imgS, facesCurFrame)
matchIndex] < 0.45:
1/1 [==========] - 0s 43ms/step





DISCUSSION

- Edge detection improves gesture clarity and recognition accuracy.
- Gamma correction balances brightness in low-light conditions.
- SafeHouse Mode effectively monitors and restricts unknown access.
- The modular structure allows easy expansion (e.g., more gestures, cloud alerts).

CONCLUSION

“SafeHomeCam” successfully integrates gesture recognition, face authentication, and alert automation to build an intelligent home safety system.
It demonstrates the potential of computer vision and AI in enhancing household security. By combining real-time detection with audio and IoT-based alerts, this system lays the foundation for smart, self-learning surveillance solutions.

FUTURE SCOPE

- Add mobile app connectivity for live monitoring.
- Implement cloud-based data storage and logging.
- Enhance recognition model using deep CNNs for higher accuracy.
- Integrate object detection to detect suspicious movements.

REFERENCES

- OpenCV Documentation: https://docs.opencv.org
- cvzone Library: https://github.com/cvzone
- Twilio API Documentation: https://www.twilio.com/docs
- face_recognition Library: https://github.com/ageitgey/face_recognition