# SAFEHOMECAM

Ayush Mangela – N127

Mheet Singh – N129

Shravan Dige – N130

Pratham Pednekar – N134

MBA Tech (Computer Engineering)  
Division E2

## 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)

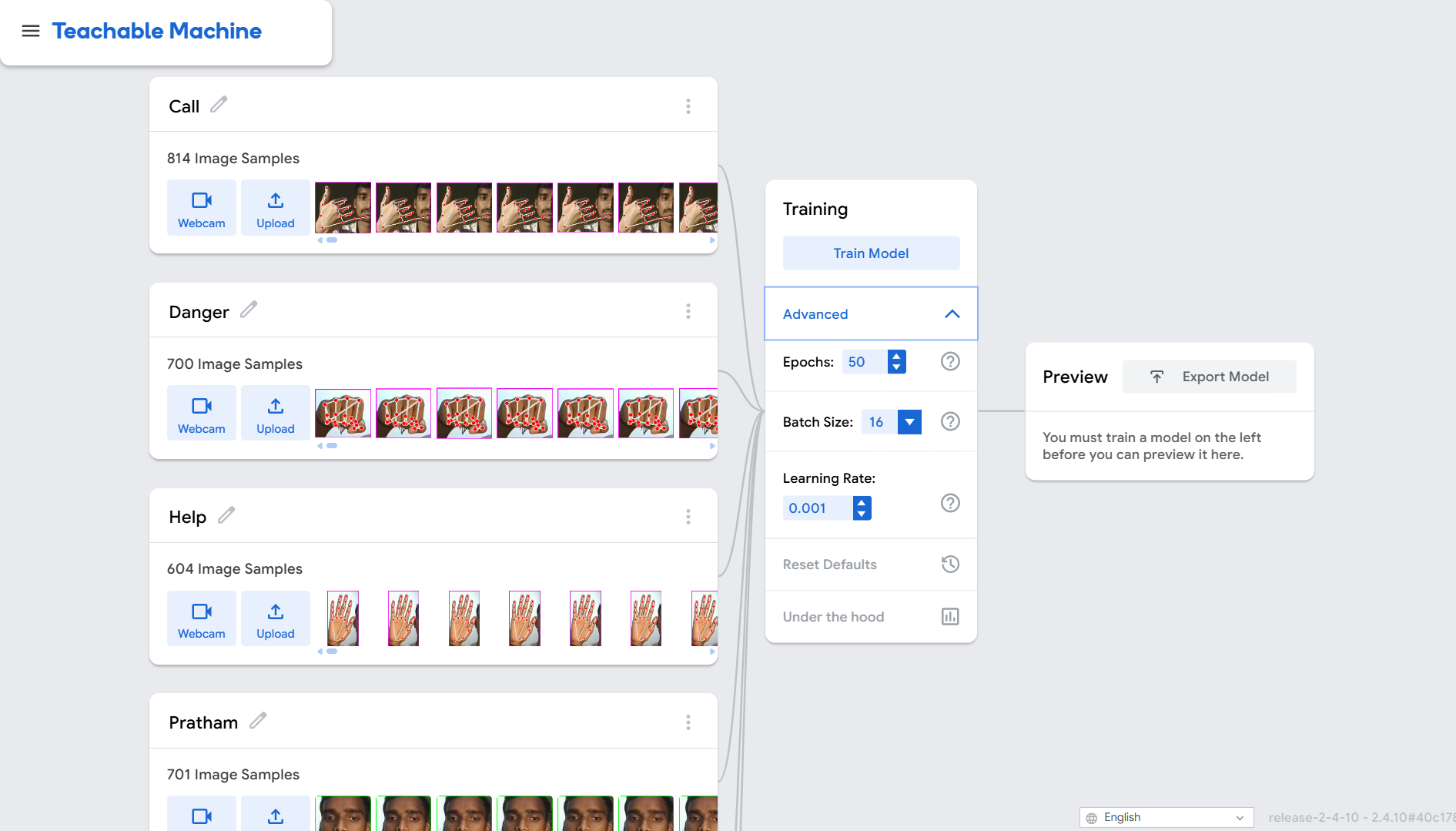
    if cv2.waitKey(1) & 0xFF == ord('q'):

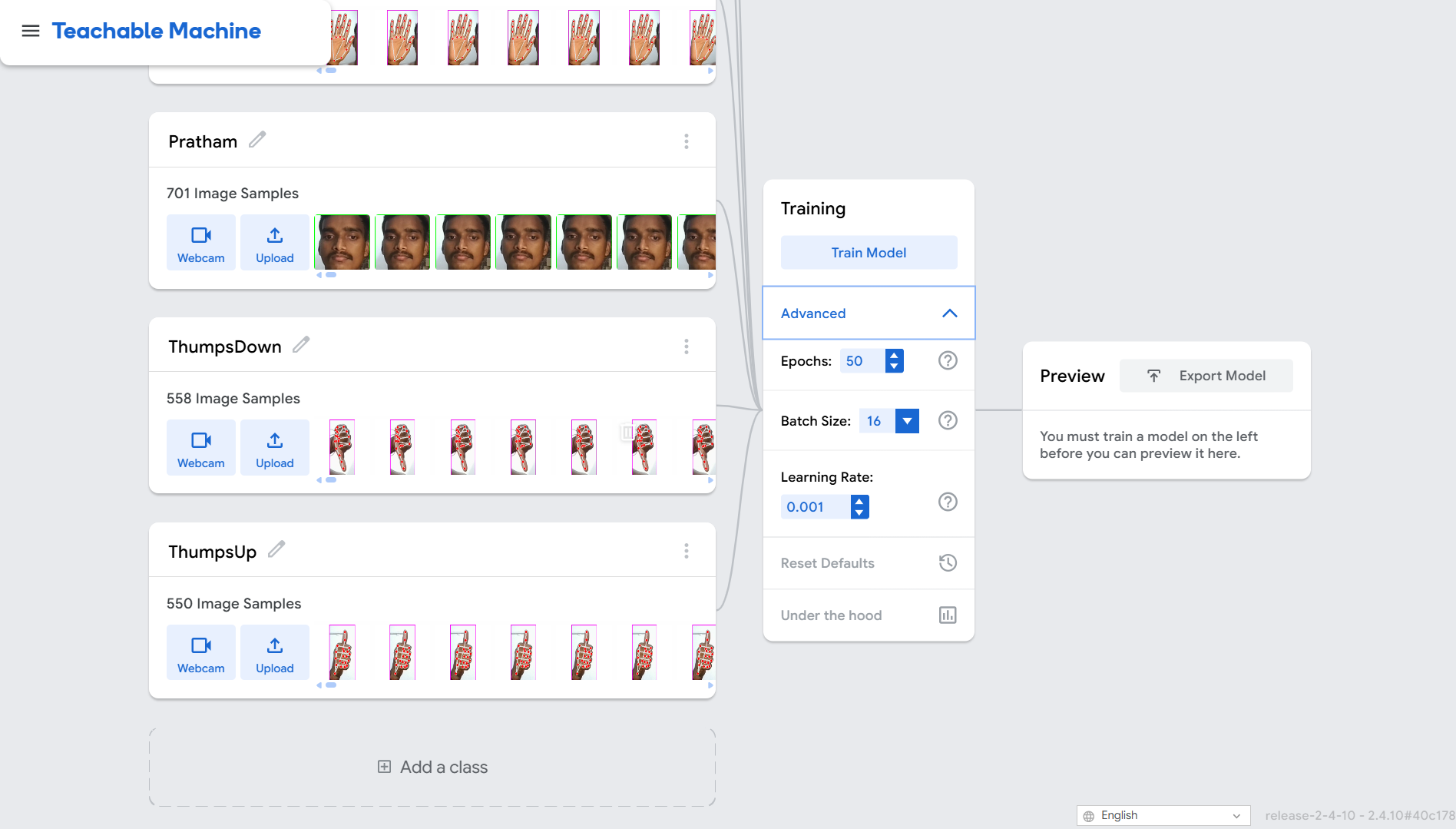
        break

cap.release()

cv2.destroyAllWindows()

Training and Testing of the model using Teachable Machine

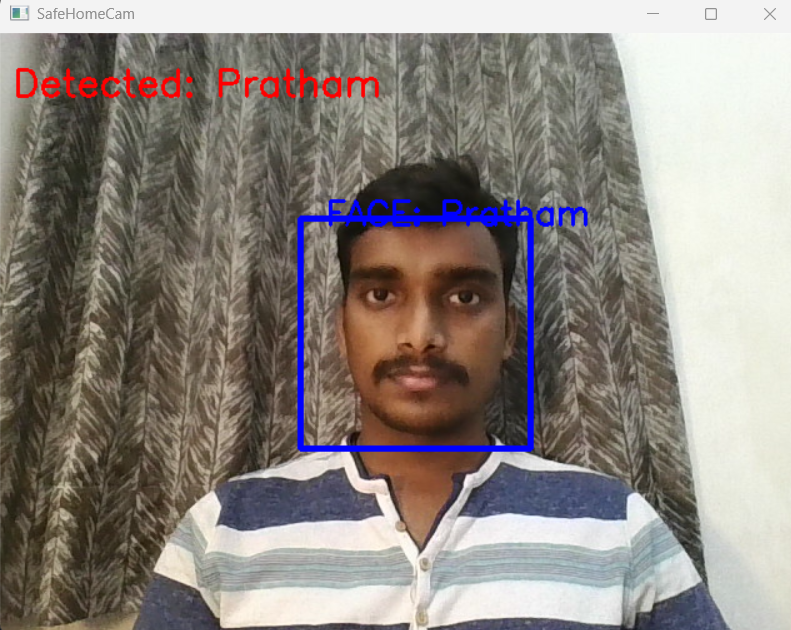


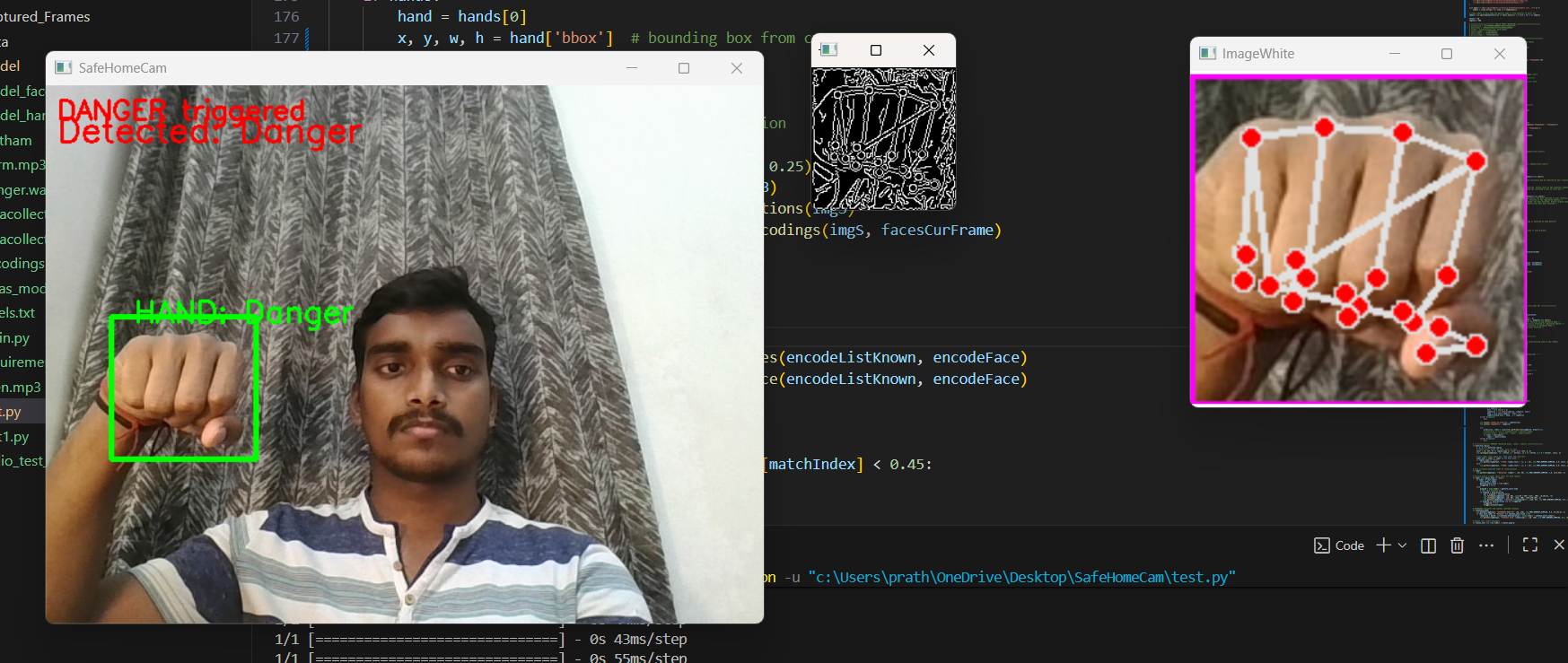
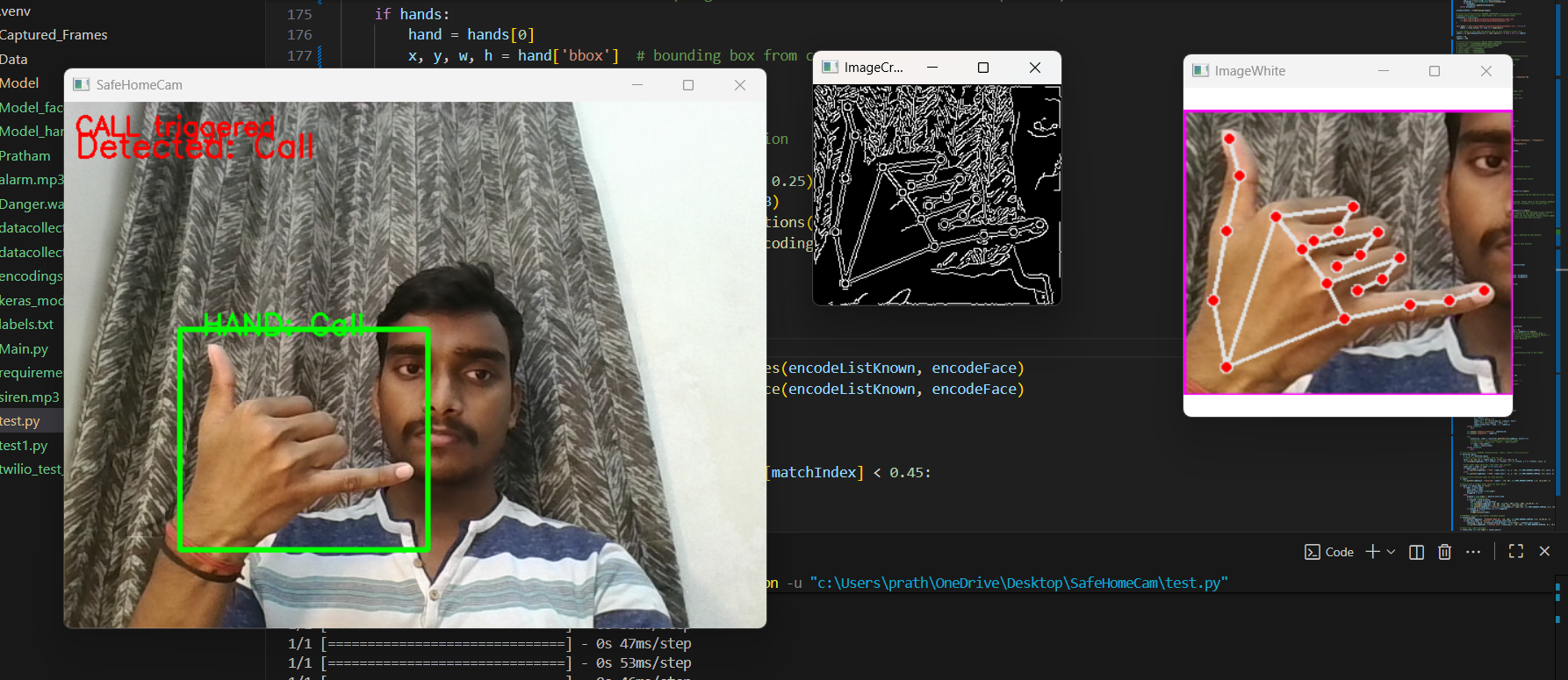
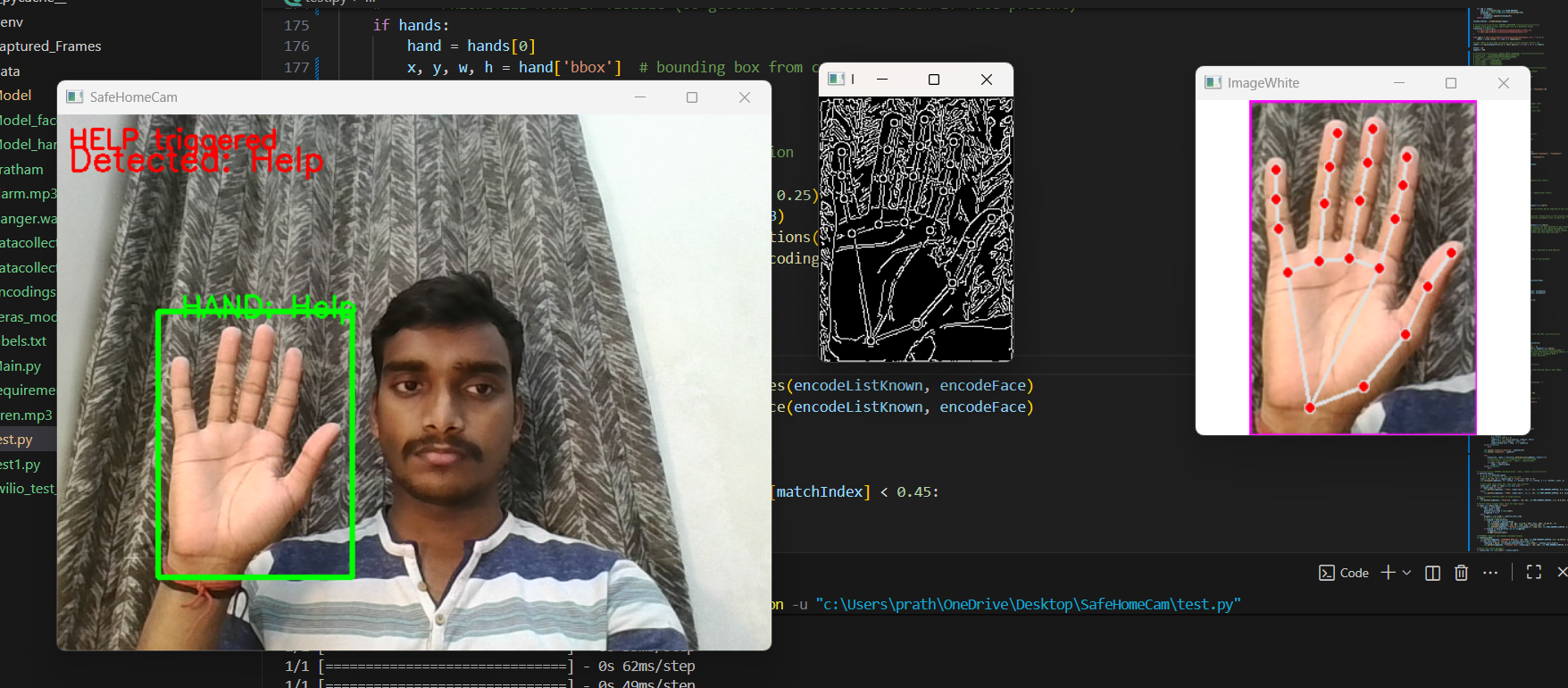


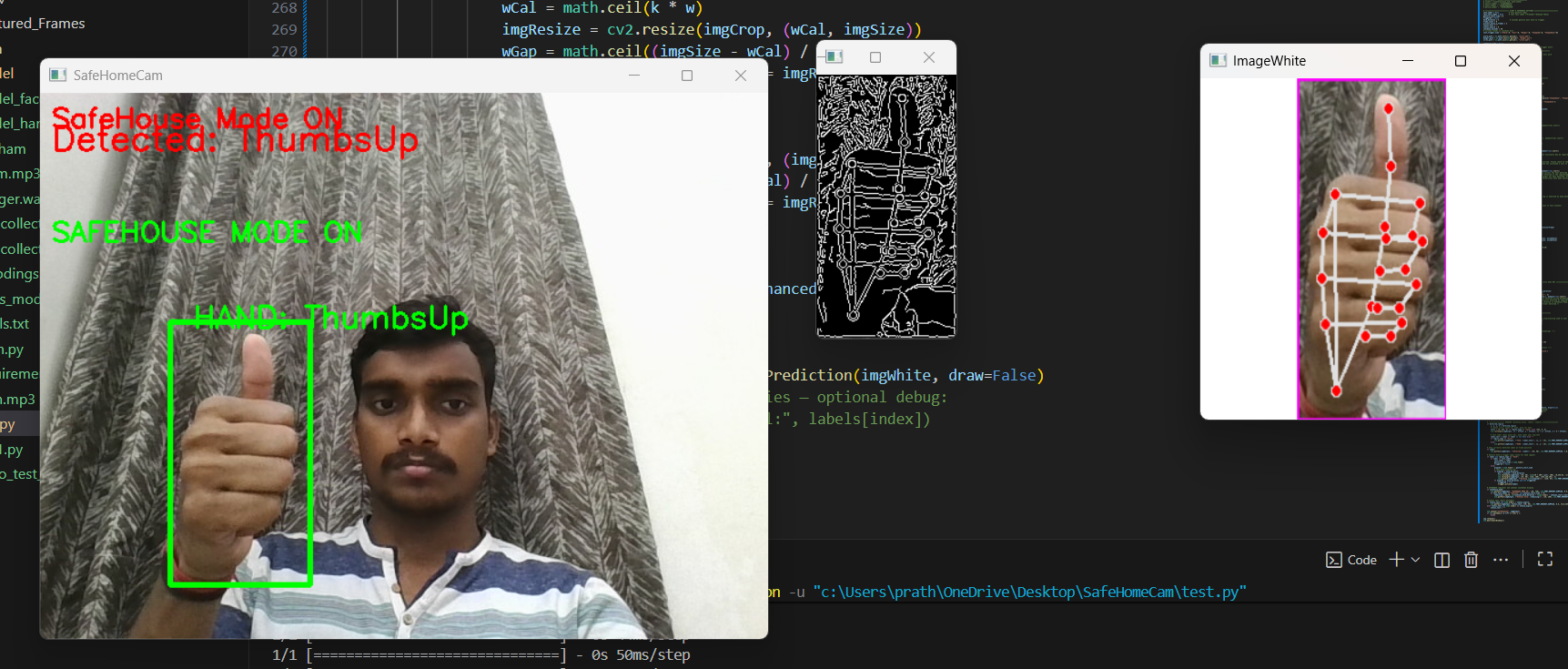
## RESULTS

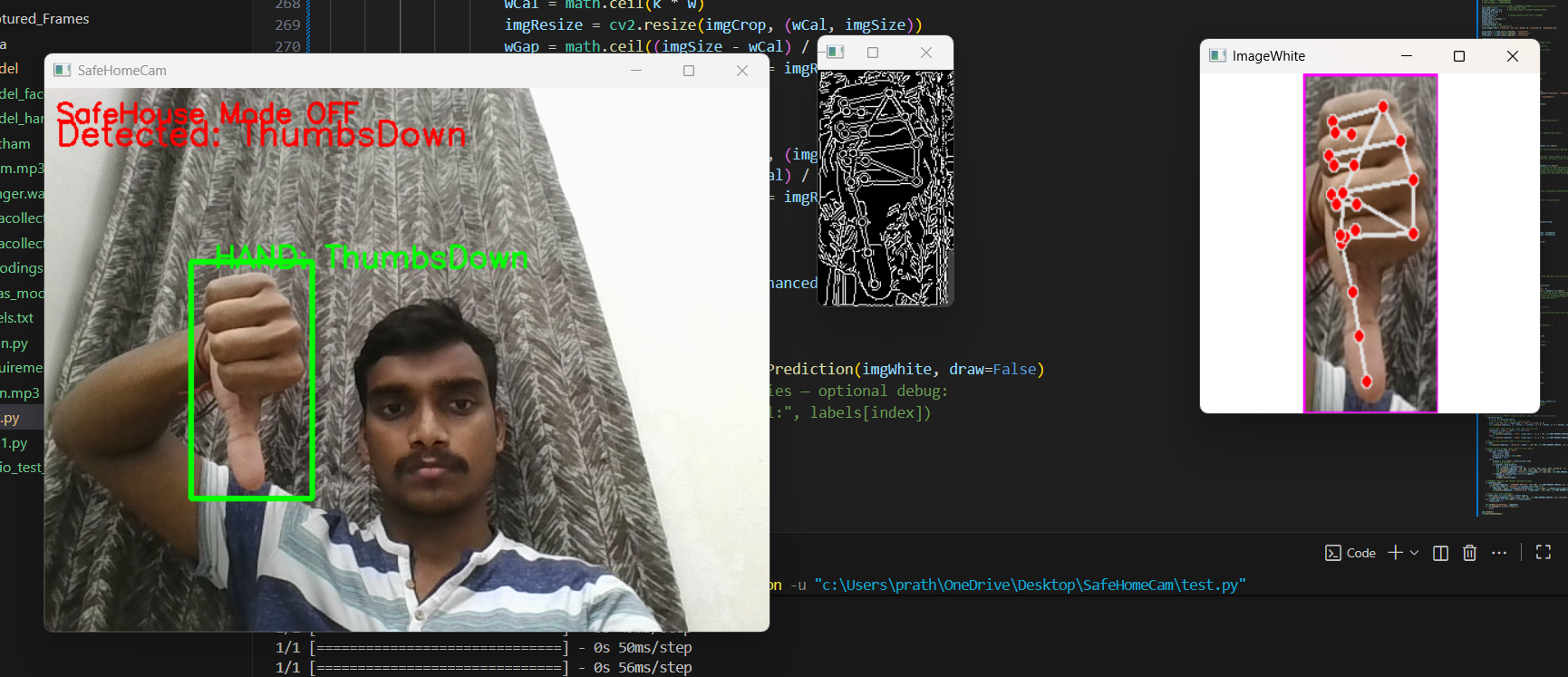
|  |  |  |
| --- | --- | --- |
| Test Case | Input | System Response |
| Known face + Thumbs Up | Gesture detected | SafeHouse Mode ON |
| Unknown face + SafeHouse Mode | Detected for 3 seconds | Siren and alert message |
| Help gesture | Detected | Alarm sound triggered |
| Thumbs Down (with owner face)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

Here is the live camera feed output :-









## 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