from imutils import face\_utils

from utils import \*

import numpy as np #for maths operations

import pyautogui as pag #for mouse controll

import imutils #for resize input frame

import dlib #for recognize face

import cv2 #accessing camera

# Thresholds and consecutive frame length for triggering the mouse action.

MOUTH\_AR\_THRESH = 0.3

MOUTH\_AR\_CONSECUTIVE\_FRAMES = 5

EYE\_AR\_THRESH = 0.20

EYE\_AR\_CONSECUTIVE\_FRAMES = 5

WINK\_AR\_DIFF\_THRESH = 0.001

WINK\_AR\_CLOSE\_THRESH = 0.2

WINK\_CONSECUTIVE\_FRAMES = 4

# Initialize the frame counters for each action as well as

# booleans used to indicate if action is performed or not

MOUTH\_COUNTER = 0

EYE\_COUNTER = 0

WINK\_COUNTER = 0

INPUT\_MODE = False

EYE\_CLICK = False

LEFT\_WINK = False

RIGHT\_WINK = False

SCROLL\_MODE = False

ANCHOR\_POINT = (0, 0)

WHITE\_COLOR = (255, 255, 255)

YELLOW\_COLOR = (0, 255, 255)

RED\_COLOR = (0, 0, 255)

GREEN\_COLOR = (0, 255, 0)

BLUE\_COLOR = (255, 0, 0)

BLACK\_COLOR = (0, 0, 0)

# Initialize Dlib's face detector (HOG-based) and then create

# the facial landmark predictor

shape\_predictor = "model/shape\_predictor\_68\_face\_landmarks.dat"

detector = dlib.get\_frontal\_face\_detector()

predictor = dlib.shape\_predictor(shape\_predictor)

# Grab the indexes of the facial landmarks for the left and

# right eye, nose and mouth respectively

(lStart, lEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS["left\_eye"]

(rStart, rEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS["right\_eye"]

(nStart, nEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS["nose"]

(mStart, mEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS["mouth"]

# Video capture

vid = cv2.VideoCapture(0)

resolution\_w = 1366

resolution\_h = 768

cam\_w = 640

cam\_h = 480

unit\_w = resolution\_w / cam\_w

unit\_h = resolution\_h / cam\_h

while True:

    print(WINK\_COUNTER,"WINK\_COUNTER")

    print(WINK\_CONSECUTIVE\_FRAMES,"WINK\_CONSECUTIVE\_FRAMES")

    # Grab the frame from the threaded video file stream, resize

    # it, and convert it to grayscale

    # channels)

    \_, frame = vid.read()

    frame = cv2.flip(frame, 1)

    frame = imutils.resize(frame, width=cam\_w, height=cam\_h)

    gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

    # Detect faces in the grayscale frame

    rects = detector(gray, 0)

    # Loop over the face detections

    if len(rects) > 0:

        rect = rects[0]

    else:

        cv2.imshow("Frame", frame)

        key = cv2.waitKey(1) & 0xFF

        continue

    # Determine the facial landmarks for the face region, then

    # convert the facial landmark (x, y)-coordinates to a NumPy

    # array

    shape = predictor(gray, rect)

    shape = face\_utils.shape\_to\_np(shape)

    # Extract the left and right eye coordinates, then use the

    # coordinates to compute the eye aspect ratio for both eyes

    mouth = shape[mStart:mEnd]

    leftEye = shape[lStart:lEnd]

    rightEye = shape[rStart:rEnd]

    nose = shape[nStart:nEnd]

    # Because I flipped the frame, left is right, right is left.

    temp = leftEye

    leftEye = rightEye

    rightEye = temp

    # Average the mouth aspect ratio together for both eyes

    mar = mouth\_aspect\_ratio(mouth)

    leftEAR = eye\_aspect\_ratio(leftEye)

    rightEAR = eye\_aspect\_ratio(rightEye)

    ear = (leftEAR + rightEAR) / 2.0

    diff\_ear = np.abs(leftEAR - rightEAR)

    nose\_point = (nose[3, 0], nose[3, 1])

    # Compute the convex hull for the left and right eye, then

    # visualize each of the eyes

    mouthHull = cv2.convexHull(mouth)

    leftEyeHull = cv2.convexHull(leftEye)

    rightEyeHull = cv2.convexHull(rightEye)

    cv2.drawContours(frame, [mouthHull], -1, YELLOW\_COLOR, 1)

    cv2.drawContours(frame, [leftEyeHull], -1, YELLOW\_COLOR, 1)

    cv2.drawContours(frame, [rightEyeHull], -1, YELLOW\_COLOR, 1)

    for (x, y) in np.concatenate((mouth, leftEye, rightEye), axis=0):

        cv2.circle(frame, (x, y), 2, GREEN\_COLOR, -1)

    # Check to see if the eye aspect ratio is below the blink

    # threshold, and if so, increment the blink frame counter

    if diff\_ear > WINK\_AR\_DIFF\_THRESH:

        if leftEAR < rightEAR:

            if leftEAR < EYE\_AR\_THRESH:

                WINK\_COUNTER += 1

                if WINK\_COUNTER > WINK\_CONSECUTIVE\_FRAMES:

                    pag.click(button='left')

                    WINK\_COUNTER = 0

        elif leftEAR > rightEAR:

            if rightEAR < EYE\_AR\_THRESH:

                WINK\_COUNTER += 1

                if WINK\_COUNTER > WINK\_CONSECUTIVE\_FRAMES:

                    pag.click(button='right')

                    WINK\_COUNTER = 0

        else:

            WINK\_COUNTER = 0

    else:

        if ear <= EYE\_AR\_THRESH:

            EYE\_COUNTER += 1

            if EYE\_COUNTER > EYE\_AR\_CONSECUTIVE\_FRAMES:

                SCROLL\_MODE = not SCROLL\_MODE

                # INPUT\_MODE = not INPUT\_MODE

                EYE\_COUNTER = 0

                # nose point to draw a bounding box around it

        else:

            EYE\_COUNTER = 0

            WINK\_COUNTER = 0

    if mar > MOUTH\_AR\_THRESH:

        MOUTH\_COUNTER += 1

        if MOUTH\_COUNTER >= MOUTH\_AR\_CONSECUTIVE\_FRAMES:

            # if the alarm is not on, turn it on

            INPUT\_MODE = not INPUT\_MODE

            # SCROLL\_MODE = not SCROLL\_MODE

            MOUTH\_COUNTER = 0

            ANCHOR\_POINT = nose\_point

    else:

        MOUTH\_COUNTER = 0

    if INPUT\_MODE:

        cv2.putText(frame, "READING INPUT!", (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, RED\_COLOR, 2)

        x, y = ANCHOR\_POINT

        nx, ny = nose\_point

        w, h = 60, 35

        multiple = 1

        cv2.rectangle(frame, (x - w, y - h), (x + w, y + h), GREEN\_COLOR, 2)

        cv2.line(frame, ANCHOR\_POINT, nose\_point, BLUE\_COLOR, 2)

        dir = direction(nose\_point, ANCHOR\_POINT, w, h)

        cv2.putText(frame, dir.upper(), (10, 90), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, RED\_COLOR, 2)

        drag = 18

        if dir == 'right':

            pag.moveRel(drag, 0)

        elif dir == 'left':

            pag.moveRel(-drag, 0)

        elif dir == 'up':

            if SCROLL\_MODE:

                pag.scroll(40)

            else:

                pag.moveRel(0, -drag)

        elif dir == 'down':

            if SCROLL\_MODE:

                pag.scroll(-40)

            else:

                pag.moveRel(0, drag)

    if SCROLL\_MODE:

        cv2.putText(frame, 'SCROLL MODE IS ON!', (10, 60), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, RED\_COLOR, 2)

    # cv2.putText(frame, "MAR: {:.2f}".format(mar), (500, 30),

    #             cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, YELLOW\_COLOR, 2)

    # cv2.putText(frame, "Right EAR: {:.2f}".format(rightEAR), (460, 80),

    #             cv2.FONT\_HERSHEY\_SIMPLqEX, 0.7, YELLOW\_COLOR, 2)

    # cv2.putText(frame, "Left EAR: {:.2f}".format(leftEAR), (460, 130),

    #             cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, YELLOW\_COLOR, 2)

    # cv2.putText(frame, "Diff EAR: {:.2f}".format(np.abs(leftEAR - rightEAR)), (460, 80),

    #             cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

    # Show the frame

    cv2.imshow("Frame", frame)

    key = cv2.waitKey(1) & 0xFF

    # If the `Esc` key was pressed, break from the loop

    if key == 27:

        break

# Do a bit of cleanup

cv2.destroyAllWindows()

vid.release()