import cv2

import mediapipe as mp

import numpy as np

import pyautogui

from filterpy.kalman import KalmanFilter

from collections import deque

import time

# Disable PyAutoGUI fail-safe

pyautogui.FAILSAFE = False

# Screen dimensions

screen\_width, screen\_height = pyautogui.size()

# Initialize MediaPipe FaceMesh

mp\_face\_mesh = mp.solutions.face\_mesh

face\_mesh = mp\_face\_mesh.FaceMesh(refine\_landmarks=True, min\_detection\_confidence=0.7, min\_tracking\_confidence=0.7)

# Kalman Filter for smoother tracking

def initialize\_kalman():

kf = KalmanFilter(dim\_x=4, dim\_z=2)

kf.F = np.array([[1, 0, 1, 0],

[0, 1, 0, 1],

[0, 0, 1, 0],

[0, 0, 0, 1]])

kf.H = np.array([[1, 0, 0, 0],

[0, 1, 0, 0]])

kf.P \*= 1000

kf.R = np.array([[3, 0],

[0, 3]])

kf.Q = np.eye(4) \* 0.05

kf.x = np.zeros((4, 1))

return kf

kf = initialize\_kalman()

# Moving average filter

smooth\_x, smooth\_y = deque(maxlen=5), deque(maxlen=5)

def smooth\_gaze(gaze\_x, gaze\_y):

smooth\_x.append(gaze\_x)

smooth\_y.append(gaze\_y)

return np.mean(smooth\_x), np.mean(smooth\_y)

# Eye landmarks for tracking

left\_pupil\_idx = 468

right\_pupil\_idx = 473

left\_eye\_indices = [33, 160, 158, 133, 153, 144]

right\_eye\_indices = [362, 385, 387, 263, 373, 380]

def eye\_aspect\_ratio(eye):

A = np.linalg.norm(eye[1] - eye[5])

B = np.linalg.norm(eye[2] - eye[4])

C = np.linalg.norm(eye[0] - eye[3])

return (A + B) / (2.0 \* C)

blink\_threshold = 0.2

left\_wink\_threshold = 0.10

right\_wink\_threshold = 0.15

zoom\_duration\_threshold = 2.0 # Duration to trigger zoom actions

blink\_start\_time = None

left\_wink\_detected = False

right\_wink\_detected = False

zoom\_in\_start\_time = None

zoom\_out\_start\_time = None

def map\_gaze\_to\_screen(gaze\_x, gaze\_y):

screen\_x = int(gaze\_x \* screen\_width)

screen\_y = int(gaze\_y \* screen\_height)

return screen\_x, screen\_y

def get\_gaze\_coordinates(frame):

global blink\_start\_time, left\_wink\_detected, right\_wink\_detected, zoom\_in\_start\_time, zoom\_out\_start\_time

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

results = face\_mesh.process(rgb\_frame)

if results is None or results.multi\_face\_landmarks is None:

return None

landmarks = results.multi\_face\_landmarks[0]

gaze\_x = (landmarks.landmark[left\_pupil\_idx].x + landmarks.landmark[right\_pupil\_idx].x) / 2

gaze\_y = (landmarks.landmark[left\_pupil\_idx].y + landmarks.landmark[right\_pupil\_idx].y) / 2

left\_eye = np.array([(landmarks.landmark[i].x, landmarks.landmark[i].y) for i in left\_eye\_indices])

right\_eye = np.array([(landmarks.landmark[i].x, landmarks.landmark[i].y) for i in right\_eye\_indices])

left\_ear = eye\_aspect\_ratio(left\_eye)

right\_ear = eye\_aspect\_ratio(right\_eye)

ear = (left\_ear + right\_ear) / 2.0

if left\_ear < left\_wink\_threshold and right\_ear > blink\_threshold:

if not left\_wink\_detected:

left\_wink\_detected = True

pyautogui.click(button='left')

else:

left\_wink\_detected = False

if right\_ear < right\_wink\_threshold and left\_ear > blink\_threshold:

if not right\_wink\_detected:

right\_wink\_detected = True

pyautogui.click(button='right')

else:

right\_wink\_detected = False

if left\_ear < blink\_threshold and right\_ear < blink\_threshold:

if blink\_start\_time is None:

blink\_start\_time = time.time()

elif time.time() - blink\_start\_time >= 1.5:

pyautogui.click()

blink\_start\_time = None

else:

blink\_start\_time = None

if right\_ear > blink\_threshold and left\_ear < blink\_threshold:

if zoom\_out\_start\_time is None:

zoom\_out\_start\_time = time.time()

elif time.time() - zoom\_out\_start\_time >= zoom\_duration\_threshold:

pyautogui.hotkey('ctrl', '-') # Zoom out

zoom\_out\_start\_time = None

else:

zoom\_out\_start\_time = None

if left\_ear > blink\_threshold and right\_ear < blink\_threshold:

if zoom\_in\_start\_time is None:

zoom\_in\_start\_time = time.time()

elif time.time() - zoom\_in\_start\_time >= zoom\_duration\_threshold:

pyautogui.hotkey('ctrl', '+') # Zoom in

zoom\_in\_start\_time = None

else:

zoom\_in\_start\_time = None

return smooth\_gaze(gaze\_x, gaze\_y)

# Start video capture

cap = cv2.VideoCapture(0)

while cap.isOpened():

ret, frame = cap.read()

if not ret:

break

frame = cv2.flip(frame, 1)

gaze\_coords = get\_gaze\_coordinates(frame)

if gaze\_coords is not None:

gaze\_x, gaze\_y = gaze\_coords

screen\_x, screen\_y = map\_gaze\_to\_screen(gaze\_x, gaze\_y)

kf.predict()

kf.update(np.array([[screen\_x], [screen\_y]]))

smoothed\_x, smoothed\_y = kf.x[0, 0], kf.x[1, 0]

pyautogui.moveTo(smoothed\_x, smoothed\_y, duration=0.01)

if smoothed\_y < screen\_height \* 0.4:

pyautogui.scroll(5)

elif smoothed\_y > screen\_height \* 0.6:

pyautogui.scroll(-5)

cv2.imshow("Eye Tracking", frame)

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

break

cap.release()

cv2.destroyAllWindows()