|  |
| --- |
| # import the necessary packages |
|  |  | from scipy.spatial import distance as dist |
|  |  | from imutils.video import FileVideoStream |
|  |  | from imutils.video import VideoStream |
|  |  | from imutils import face\_utils |
|  |  | import numpy as np |
|  |  | import argparse |
|  |  | import imutils |
|  |  | import time |
|  |  | import dlib |
|  |  | import cv2 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | def eye\_aspect\_ratio(eye): |
|  |  | A = dist.euclidean(eye[1], eye[5]) |
|  |  | B = dist.euclidean(eye[2], eye[4]) |
|  |  | C = dist.euclidean(eye[0], eye[3]) |
|  |  |  |
|  |  | #Compute eye aspect ratio |
|  |  | ear = (A+B)/(2\*C) |
|  |  |  |
|  |  | return ear |
|  |  |  |
|  |  | def mouth\_aspect\_ratio(mouth): |
|  |  | A = dist.euclidean(mouth[2],mouth[10]) |
|  |  | B = dist.euclidean(mouth[3],mouth[9]) |
|  |  | C = dist.euclidean(mouth[4],mouth[8]) |
|  |  |  |
|  |  | mar = (A+B+C)/3 |
|  |  |  |
|  |  | return mar |
|  |  |  |
|  |  |  |
|  |  | ap = argparse.ArgumentParser() |
|  |  | ap.add\_argument("-p", "--shape-predictor", required=True, |
|  |  | help="path to facial landmark predictor") |
|  |  | ap.add\_argument("-v", "--video", type=str, default="", |
|  |  | help="path to input video file") |
|  |  | args = vars(ap.parse\_args()) |
|  |  |  |
|  |  | EYE\_AR\_THRESH = 0.3 #threshold for blink |
|  |  | EYE\_AR\_CONSEC\_FRAMES = 10 #consecutive considered true |
|  |  | sleep\_flag = -1 |
|  |  |  |
|  |  | counter = 0 |
|  |  | total = 0 |
|  |  |  |
|  |  | print("[INFO] loading facial landmark predictor...") |
|  |  | detector = dlib.get\_frontal\_face\_detector() |
|  |  | predictor = dlib.shape\_predictor(args["shape\_predictor"]) |
|  |  |  |
|  |  | (lStart, lEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS["left\_eye"] |
|  |  | (rStart, rEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS["right\_eye"] |
|  |  | (mStart, mEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS["mouth"] |
|  |  |  |
|  |  | print("[INFO] starting video stream thread...") |
|  |  | vs = FileVideoStream(args["video"]).start() |
|  |  | fileStream = True |
|  |  | vs = VideoStream(src=0).start() |
|  |  | time.sleep(1.0) |
|  |  | start\_time = time.time() |
|  |  | elapsed\_time = start\_time |
|  |  |  |
|  |  | while True: |
|  |  | # if fileStream and not vs.more(): |
|  |  | # break |
|  |  |  |
|  |  | frame = vs.read() |
|  |  | frame = imutils.resize(frame, width = 450) |
|  |  | gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY) |
|  |  |  |
|  |  | rects = detector(gray,0) #dlib’s built-in face detector. |
|  |  |  |
|  |  | for rect in rects: |
|  |  | shape = predictor(gray, rect) |
|  |  | shape = face\_utils.shape\_to\_np(shape) |
|  |  | leftEye = shape[lStart:lEnd] |
|  |  | rightEye = shape[rStart:rEnd] |
|  |  | leftEAR = eye\_aspect\_ratio(leftEye) |
|  |  | rightEAR = eye\_aspect\_ratio(rightEye) |
|  |  | mouth = shape[mStart: mEnd] |
|  |  | mouthEAR = mouth\_aspect\_ratio(mouth) |
|  |  |  |
|  |  | ear = (leftEAR + rightEAR) / 2.0 |
|  |  | leftEyeHull = cv2.convexHull(leftEye) |
|  |  | rightEyeHull = cv2.convexHull(rightEye) |
|  |  | mouthHull = cv2.convexHull(mouth) |
|  |  | cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1) |
|  |  | cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1) |
|  |  | cv2.drawContours(frame, [mouthHull], -1, (0, 255, 0), 1) |
|  |  |  |
|  |  | if mouthEAR > 25: |
|  |  | print("You are yawning") |
|  |  | # else: |
|  |  | # print("You are working") |
|  |  | if ear < EYE\_AR\_THRESH: |
|  |  | counter += 1 |
|  |  |  |
|  |  | if counter >= EYE\_AR\_CONSEC\_FRAMES: |
|  |  | if sleep\_flag < 0: |
|  |  | print("You are sleeping") |
|  |  | sleep\_flag = 1 |
|  |  | # if total > 0: |
|  |  | # elapsed\_time = time.time() - start\_time |
|  |  | # if elapsed\_time > 10: |
|  |  | # print("You are sleeping dear") |
|  |  | # start\_time = time.time() |
|  |  | # else: |
|  |  | # print("You are working") |
|  |  |  |
|  |  | else: |
|  |  | counter = 0 |
|  |  |  |
|  |  | cv2.putText(frame, "Blinks: {}".format(total), (10, 30), |
|  |  | cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2) |
|  |  | cv2.putText(frame, "EAR: {:.2f}".format(ear), (300, 30), |
|  |  | cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2) |
|  |  |  |
|  |  | cv2.imshow("Frame", frame) |
|  |  | key = cv2.waitKey(1) & 0xFF |
|  |  | if key == ord("q"): |
|  |  | break |
|  |  |  |
|  |  | # do a bit of clean   |  | | --- | | cv2.destroyAllWindows() | |  |  | vs.stop() |   Vs.stop()  Eyedetect.py   |  | | --- | | from imutils.video import VideoStream | |  |  | from imutils import face\_utils | |  |  | import argparse | |  |  | import imutils | |  |  | import time | |  |  | import dlib | |  |  | import cv2 | |  |  |  | |  |  |  | |  |  | ap = argparse.ArgumentParser() | |  |  | ap.add\_argument("-p", "--shape-predictor", required=True, | |  |  | help="path to facial landmark predictor") | |  |  | ap.add\_argument("-r", "--picamera", type=int, default=-1, | |  |  | help="whether or not the Raspberry Pi camera should be used") | |  |  | args = vars(ap.parse\_args()) | |  |  |  | |  |  | print("[INFO] loading facial landmark predictor...") | |  |  | detector = dlib.get\_frontal\_face\_detector() | |  |  | predictor = dlib.shape\_predictor(args["shape\_predictor"]) | |  |  |  | |  |  | print("[INFO] camera sensor warming up...") | |  |  | vs = VideoStream(usePiCamera=args["picamera"] > 0).start() | |  |  | time.sleep(2.0) | |  |  |  | |  |  | while True: | |  |  |  | |  |  | frame = vs.read() | |  |  | frame = imutils.resize(frame, width=600) | |  |  | gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY) | |  |  |  | |  |  |  | |  |  | rects = detector(gray, 0) | |  |  | for (i,rect) in enumerate(rects): | |  |  |  | |  |  | shape = predictor(gray, rect) | |  |  | shape = face\_utils.shape\_to\_np(shape) | |  |  | for (name, (i,j)) in face\_utils.FACIAL\_LANDMARKS\_IDXS.items(): | |  |  | for (x, y) in shape[36:48]: | |  |  | cv2.circle(frame, (x, y), 1, (0, 0, 255), 0) | |  |  | for (x, y) in shape[48:68]: | |  |  | cv2.circle(frame, (x, y), 1, (0, 0, 255), 0) | |  |  | cv2.imshow("Frame", frame) | |  |  | key = cv2.waitKey(1) & 0xFF | |  |  |  | |  |  | if key == ord("q"): | |  |  | break | |  |  |  | |  |  | cv2.destroyAllWindows() | |  |  | vs.stop() | |