**(RTDDAM)Real-Time Driver Drowsiness Alert Mechanism**

**import** json  
**import** os  
**import** ssl  
**from** time **import** sleep, time  
  
**import** cv2  
**import** paho.mqtt.client **as** MQTT  
**import** RPi.GPIO **as** GPIO  
**from** imutils **import** face\_utils, resize  
**from** imutils.video **import** VideoStream  
**from** numpy.linalg **import** norm  
**from** serial **import** Serial  
  
**import** dlib  
  
*# define two constants, one for the eye aspect ratio to indicate blink and then a second constant for the number of consecutive frames the eye must be below the threshold for to set off the alarm ------*EYE\_AR\_THRESH = 0.3  
EYE\_AR\_CONSEC\_FRAMES = 5  
BP\_THRESH = 60  
COUNTER = 0  
CHECK\_BPM\_INTERVAL = 50  
  
*# serail port connection wtih arduino*port = Serial(**"/dev/rfcomm0"**, 9600, timeout=2)  
  
*# values for connecting to AWS IoT*host = **"ayi4kdvrrvrmp-ats.iot.us-east-2.amazonaws.com"**certPath = os.path.dirname(os.path.abspath(\_\_file\_\_))  
rootCAPath = certPath + **"/root-CA.crt"**privateKeyPath = certPath + **"/RTDDAM.private.key"**certificatePath = certPath + **"/RTDDAM.cert.pem"**clientId = **"RTDDAM-master"**topic = **"buzzer-record"**mqtt = None  
  
*# compute and return the euclidean distance between the two points***def** euclidean\_dist(ptA, ptB):  
 **return** norm(ptA - ptB)  
  
*# compute the euclidean distances between the two sets of vertical eye landmarks (x, y)-coordinates***def** eye\_aspect\_ratio(eye):  
 A = euclidean\_dist(eye[1], eye[5])  
 B = euclidean\_dist(eye[2], eye[4])  
  
 *# compute the euclidean distance between the horizontal eye landmark (x, y)-coordinates* C = euclidean\_dist(eye[0], eye[3])  
  
 *# compute and return the eye aspect ratio* **return** ((A + B) / (2.0 \* C))  
  
*# rings buzzer and transmits buzzer data to AWS IoT***def** ringBuzzer(reason):  
  
 *# ringing buzzer* GPIO.setmode(GPIO.BCM)  
 GPIO.setup(23, GPIO.OUT)  
 GPIO.output(23, True)  
  
 *# transmitting buzzer data to AWS IoT* buzzerData = {}  
 buzzerData[**'timestamp'**] = time()  
 buzzerData[**'buzzer-state'**] = reason + **" ALARM!"** buzzerDataJson = json.dumps(buzzerData)  
 mqtt.publish(topic, buzzerDataJson, qos=0)  
 **print**(**"Buzzer Data Sent : "** + buzzerDataJson)  
 sleep(5)  
  
 *# stopping buzzer* GPIO.cleanup()  
  
*# detects eyes ear factor***def** detectEyes(vs, detector, predictor, lStart, lEnd, rStart, rEnd):  
 **global** COUNTER  
  
 *# grab the frame from the threaded video file stream, resize it, and convert it to grayscale channels* frame = resize(vs.read(), width=500)  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 *# detect faces in the grayscale frame* rects = detector.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30), flags=cv2.CASCADE\_SCALE\_IMAGE)  
  
 *# loop over the face detections* **for** (x, y, w, h) **in** rects:  
  
 *# construct a dlib rectangle object from the Haar cascade bounding box* rect = dlib.rectangle(int(x), int(y), int(x + w), int(y + h))  
  
 *# determine the facial landmarks for the face region, then convert the facial landmark (x, y)-coordinates to a NumPy array* shape = face\_utils.shape\_to\_np(predictor(gray, rect))  
  
 *# extract the left and right eye coordinates, then use the coordinates to compute the eye aspect ratio for both eyes* leftEye = shape[lStart:lEnd]  
 rightEye = shape[rStart:rEnd]  
 leftEYE = eye\_aspect\_ratio(leftEye)  
 rightEYE = eye\_aspect\_ratio(rightEye)  
  
 *# average the eye aspect ratio together for both eyes* eye = (leftEYE + rightEYE) / 2.0  
  
 *# compute the convex hull for the left and right eye, then visualize each of the eyes* leftEyeHull = cv2.convexHull(leftEye)  
 rightEyeHull = cv2.convexHull(rightEye)  
 cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)  
 cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)  
  
 *# check to see if the eye aspect ratio is below the blink threshold, and if so, increment the blink frame counter* **if** eye < EYE\_AR\_THRESH:  
 COUNTER += 1  
  
 *# if the eyes were closed for a sufficient number of frames, then sound the alarm* **if** COUNTER >= EYE\_AR\_CONSEC\_FRAMES:  
  
 *# draw an alarm on the frame* cv2.putText(frame, **"DROWSINESS ALERT!"**, (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)  
  
 *# buzzer* ringBuzzer(**"EYES"**)  
  
 *# otherwise, the eye aspect ratio is not below the blink threshold, so reset the counter* **else**:  
 COUNTER = 0  
  
 *# draw the computed eye aspect ratio on the frame to help with debugging and setting the correct eye aspect ratio thresholds and frame counters* cv2.putText(frame, **"EYE: {:.3f}"**.format(eye), (300, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)  
  
 *# show the frame* cv2.imshow(**"RTDDAM"**, frame)  
  
*# detects BPM pulse***def** detectBPM():  
 **try**:  
 BPM = int(port.readline())  
 **if** BPM < BP\_THRESH:  
 ringBuzzer(**"BPM "** + str(BPM))  
   
 **except**:  
 **pass***# configres and conects to AWS IoT***def** connectToAWSIoT():  
 **global** mqtt  
  
 *# create mqtt client* mqtt = MQTT.Client()  
 mqtt.tls\_set(rootCAPath, certfile=certificatePath, keyfile=privateKeyPath, cert\_reqs=ssl.CERT\_REQUIRED, tls\_version=ssl.PROTOCOL\_TLSv1\_2, ciphers=None)  
  
 *# connecting to AWS IoT* mqtt.connect(host, 8883, keepalive=60)  
 mqtt.loop\_start()  
  
*# main function***def** main():  
   
 *# check to see if we are using GPIO/TrafficHat as an alarm if alarm  
 #gp = GPIO()* **print**(**"[INFO] using GPIO alarm..."**)  
  
 *# load OpenCV's Haar cascade for face detection (which is faster than dlib's built-in HOG detector, but less accurate), then create the facial landmark predictor* **print**(**"[INFO] loading facial landmark predictor..."**)  
 detector = cv2.CascadeClassifier(**"haarcascade\_frontalface\_default.xml"**)  
 predictor = dlib.shape\_predictor(**"shape\_predictor\_68\_face\_landmarks.dat"**)  
  
 *# grab the indexes of the facial landmarks for the left and right eye, respectively* (lStart, lEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS[**"left\_eye"**]  
 (rStart, rEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS[**"right\_eye"**]  
  
 *# start the video stream thread* **print**(**"[INFO] starting video stream thread..."**)  
 vs = VideoStream(usePiCamera=True).start()  
 sleep(1)  
   
 *# connecting to AWS IoT* **print**(**"[INFO] connecting to AWS IoT..."**)  
 connectToAWSIoT()  
  
 *# loop over frames from the video stream* checkIntervalBPMCount = 0  
 **while** True:  
  
 *# detecting eyes* detectEyes(vs, detector, predictor, lStart, lEnd, rStart, rEnd)  
  
 *# checking if ESC key was pressed* key = cv2.waitKey(1)  
 **if** key == 27:  
 **break** *# detecting BPM after a specific interval* **if** checkIntervalBPMCount == CHECK\_BPM\_INTERVAL:  
 detectBPM()  
 checkIntervalBPMCount = 0  
 checkIntervalBPMCount += 1  
  
 *# disconnecting from AWS IoT* mqtt.disconnect()  
  
 *# clearing frames and stopping video stream* cv2.destroyAllWindows()  
 vs.stop()  
  
*# main function call***if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()