**Machine Learning Internship Session 6**

Drowsiness Detection - Coding Sheet

***Python is a case sensitive language and proper indentation should be followed while programming***

```
# import the necessary packages
from imutils.video import VideoStream
from imutils import face_utils
import numpy as np
import argparse
import imutils
import time
import dlib
import cv2
import serial
MCData = serial.Serial(port = "COM15", baudrate=115200,bytesize=8, timeout=2,
stopbits=serial.STOPBITS_ONE)


def euclidean_dist(ptA, ptB):
    # compute and return the euclidean distance between the two
    # points
    return np.linalg.norm(ptA - ptB)


def eye_aspect_ratio(eye):
    # compute the euclidean distances between the two sets of
    # vertical eye landmarks (x, y)-coordinates
    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 the eye aspect ratio
```



www.barola.org | Copyrights @ Barola

```
        ear = (A + B) / (2.0 * C)


        # return the eye aspect ratio
        return ear
def getSeriAl():
    time.sleep(2)
    global DaTa_r
    if(MCData.in_waiting >0):
        line = MCData.readline()
        readdat=line.decode()


        print(readdat)
        if(readdat == "Ready\r\n"):
            print("Device Online")
# loop over frames from the video stream
getSeriAl()
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-c", "--cascade", required=False,default='haarcascade_frontalface_default.xml',
        help = "path to where the face cascade resides")
ap.add_argument("-p", "--shape-predictor", required=False,default='shape_predictor_face_landmarks.dat',
        help="path to facial landmark predictor")
ap.add_argument("-a", "--alarm", type=int, default=0,
        help="boolean used to indicate if TraffHat should be used")
args = vars(ap.parse_args())


# 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
```

```
# initialize the frame counter as well as a boolean used to

COUNTER = 0



# 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(args["cascade"])
predictor = dlib.shape_predictor(args["shape_predictor"])

# 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(src=1).start()

time.sleep(1.0)




while True:
    # grab the frame from the threaded video file stream, resize
    # it, and convert it to grayscale
    # channels)
    frame = vs.read()
    frame = imutils.resize(frame, width=450)
```


www.barola.org | Copyrights @ Barola

```
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 = 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
    leftEye = shape[lStart:lEnd]
    rightEye = shape[rStart:rEnd]
    leftEAR = eye_aspect_ratio(leftEye)
    rightEAR = eye_aspect_ratio(rightEye)

    # average the eye aspect ratio together for both eyes
    ear = (leftEAR + rightEAR) / 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 ear < 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)
        MCData.write(("1").encode('utf-8'))

# otherwise, the eye aspect ratio is not below the blink
# threshold, so reset the counter and alarm
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, "VAL: {:.3f}".format(ear), (300, 30),
    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
```


www.barola.org | Copyrights @ Barola

```
        # show the frame
        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF

        # if the `q` key was pressed, break from the loop
        if key == ord("q"):
            break

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```

End of Document