PYTHON

```
# 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
import winsound
arduinoData = serial.Serial('COM4', 115200)
# 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
EYE_AR_THRESH = 0.38
EYE_AR_CONSEC_FRAMES =4
DaTa_r=""
def getSeriAl():
    global DaTa_r
    if(arduinoData.in_waiting >0):
        line = arduinoData.readline()
        readdat=line.decode()
```

```
        print(readdat)
        if(readdat == "Ready\r\n"):
            print("device online")
            DaTa_r="run"
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
        ear = (A + B) / (2.0 * C)
 # return the eye aspect ratio
      return ear
 # initialize the frame counter

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("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(src=0).start()
# vs = VideoStream(usePiCamera=True).start()
print("[INFO] Please Wait for hardware booting...")
time.sleep(1.0)
found=set()
# loop over frames from the video stream
while True:
    # grab the frame from the threaded video file stream, resize
    # it, and convert it to grayscale
    # channels)
    getSeriAl()
    frame = vs.read()
    frame = imutils.resize(frame, width=450)
    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:
        #print('detected')
if(COUNTER>EYE_AR_CONSEC_FRAMES):
            if ("alert" not in found and DaTa_r=="run"):
                print("alert")
                arduinoData.write(('alert').encode())
                time.sleep(0.3)
                found.clear()
                found.add("alert")
                cv2.putText(frame, "DROWSINESS ALERT!", (10, 30),
                   cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
        elif(COUNTER>2):
            winsound.PlaySound("*", winsound.SND_ALIAS)
            COUNTER+=1
        else:
            COUNTER+=1
   # otherwise, the eye aspect ratio is not below the blink
    # threshold, so reset the counter and alert
    else:
```

```
                if ("run" not in found and DaTa_r=="run") :
                    print("run")
                    arduinoData.write(('run').encode())
                    time.sleep(0.3)
                    found.clear()
                    found.add("run")
                COUNTER = 0
                #print("running")


        # 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)
          # 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()
vs.stop()
```

ARDUINO IDE

```
#include<LiquidCrystal.>
LiquidCrystal lcd(13, 12, 11, 10, 9, 8);
#define python Serial
#define splash splash1
#define buz 6
String IncomingData ="";
float ti;
void setup()
{
python.begin(115200);
LcDSet();
splash(0, "Initializing");
splash(1, "Python");
pinMode(buz,OUTPUT);
digitalWrite(buz,LOW);
delay(3000);
lcd.clear();
python.println("Ready");


}
void LcDSet() {
lcd.begin(16, 2);
splash(0, "Drowsiness");
splash(1, "TX");
delay(3000);
lcd.clear();
}
void loop()
{
// ti += 0.5;
while (python.available())
{
IncomingData = python.readString();
delayMicroseconds(5);
```

```
}
If
(IncomingData.length()>0) {
if (IncomingData == "run") {

splash(1, "Run");
python.println("Running");
}
if (IncomingData == "alert") {
splash(1, "alert");
python.println("alert");
}
IncomingData = "";
}
delay(200);
}

// SPLASH

void splash1( int row, String txt)
{
int curs = (17 - txt.length()) / 2;
lcd.setCursor(0, row);
lcd.print("----------------");
lcd.setCursor(curs, row);
lcd.print(txt);
delay(300);
}
```