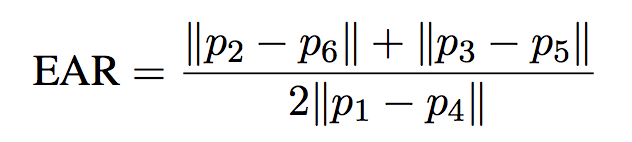
**WORKING**

A program was developed to identify the driver's drowsiness based on real-time camera image and image processing techniques, and this program makes warning through alarm, when it detects drowsiness driving.

Extracting face and eye region: Using the Histogram Of Gradients face pattern, to find the face from the Grayscaled-HOG-input-image .Use the Face Landmark Estimation algorithm to locate the landmarks on the face.

## Drowsiness detection method: Each eye is represented by 6 (x, y)-coordinatesThe Eye Aspect Ratio (EAR) is calculated using six (x, y) coordinates for the detected eye. The EAR equation

[](https://user-images.githubusercontent.com/36785390/52702578-cb71c280-2fbf-11e9-9a06-d4434250d622.png)

* Calculated EAR
* The calculated EAR will have a value more than zero when the eyes are open, and a value close to zero when the eyes are closed.
* This program has set a 50% value from the average EAR value to the threshold value.

1) measures the average EAR value when the eyes are open. 2) measures the average EAR value when the driver is closing his eyes

3) sets the threshold using the above two results.

* The computed EAR has a nonzero value when you open your eyes and a value close to zero when you close your eyes. If you set a certain threshold to the threshold (the threshold used to determine drowsy operation), you can detect that the driver is drowsy by checking whether the EAR value is smaller than that value.
* In addition, since it is not necessary to examine both eyes separately to determine drowsy driving, the average of EAR values ​​of each eye was used.
* Threshold value is set to 50% of EAR value when eyes are opened. If it's smaller (when the eye size is smaller), the driver thinks you're sleepy and cares about whether you're sleepy, so the alarm sounds even if you're not in full sleep.
* To apply this algorithm, we applied three steps:

1) Determine average EAR value when driver opens eyes

2) Determine average EAR value when driver closes eyes

3) EAR value that is 50% of eyes open using the above two values.

if EAR < threshold for 20 frame then going alarm off.

The drowsiness phase is divided according to the time when the eyes are closed and the time the eyes were opened before the drowsiness operation.To distinguish drowsiness level, we used K-Nearest Neighbor(KNN) supervised learning algorithm.

**REQUIREMENTS**

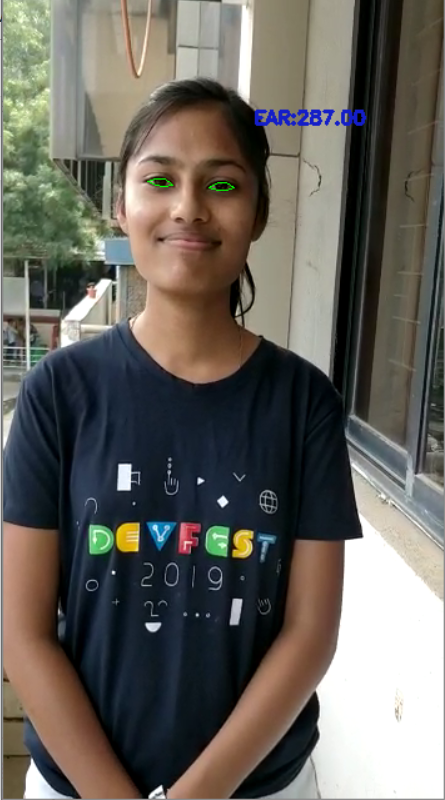
**Hardward requirements :**

* Laptop
* A camera with good video quality

**Software requirements:**

* Operating system
* Python IDE
* Jupyter Notebook

**SNAPSHOTS**

** **

Eyes are open Eyes are closed

**6.Code**

//KNN Algorithm

import numpy as np

import cv2

import matplotlib.pyplot as plt

from sklearn import metrics

knn = cv2.ml.KNearest\_create()

def start(sample\_size=25) :

train\_data = generate\_data(sample\_size)

labels = classify\_label(train\_data)

power, nomal, short = binding\_label(train\_data, labels)

print("Return true if training is successful :", knn.train(train\_data, cv2.ml.ROW\_SAMPLE, labels))

return power, nomal, short

#'num\_samples' is number of data points to create

#'num\_features' means the number of features at each data point (in this case, x-y conrdination values)

def generate\_data(num\_samples, num\_features = 2) :

"""randomly generates a number of data points"""

data\_size = (num\_samples, num\_features)

data = np.random.randint(0,40, size = data\_size)

return data.astype(np.float32)

#I determined the drowsiness-driving-risk-level based on the time which can prevent driving accident.

def classify\_label(train\_data):

labels = []

for data in train\_data :

if data[1] < data[0]-15 :

labels.append(2)

elif data[1] >= (data[0]/2 + 15) :

labels.append(0)

else :

labels.append(1)

return np.array(labels)

def binding\_label(train\_data, labels) :

power = train\_data[labels==0]

nomal = train\_data[labels==1]

short = train\_data[labels==2]

return power, nomal, short

//Main Code

import numpy as np

import imutils

import time

import timeit

import dlib

import cv2

import matplotlib.pyplot as plt

from scipy.spatial import distance as dist

from imutils.video import VideoStream

from imutils import face\_utils

from threading import Thread

from threading import Timer

from check\_cam\_fps import check\_fps

import make\_train\_data as mtd

import light\_remover as lr

import ringing\_alarm as alarm

def yawn(mouth):

return ((dist.euclidean(mouth[2], mouth[10])+dist.euclidean(mouth[4], mouth[8]))/(2\*dist.euclidean(mouth[0], mouth[6])))

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])

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

return ear

def init\_open\_ear() :

time.sleep(5)

print("open init time sleep")

ear\_list = []

th\_message1 = Thread(target = init\_message)

th\_message1.deamon = True

th\_message1.start()

for i in range(7) :

ear\_list.append(both\_ear)

time.sleep(1)

global OPEN\_EAR

OPEN\_EAR = sum(ear\_list) / len(ear\_list)

print("open list =", ear\_list, "\nOPEN\_EAR =", OPEN\_EAR, "\n")

def init\_close\_ear() :

time.sleep(2)

th\_open.join()

time.sleep(5)

print("close init time sleep")

ear\_list = []

th\_message2 = Thread(target = init\_message)

th\_message2.deamon = True

th\_message2.start()

time.sleep(1)

for i in range(7) :

ear\_list.append(both\_ear)

time.sleep(1)

CLOSE\_EAR = sum(ear\_list) / len(ear\_list)

global EAR\_THRESH

EAR\_THRESH = (((OPEN\_EAR - CLOSE\_EAR) / 2) + CLOSE\_EAR) #EAR\_THRESH means 50% of the being opened eyes state

print("close list =", ear\_list, "\nCLOSE\_EAR =", CLOSE\_EAR, "\n")

print("The last EAR\_THRESH's value :",EAR\_THRESH, "\n")

def init\_message() :

print("init\_message")

#1. Variables for checking EAR.

OPEN\_EAR = 0 #For init\_open\_ear()

EAR\_THRESH = 0 #Threashold value

#2. Variables for detecting if user is asleep.

EAR\_CONSEC\_FRAMES = 20

COUNTER = 0 #Frames counter.

#3. When the alarm rings, measure the time eyes are being closed.

closed\_eyes\_time = [] #The time eyes were being offed.

TIMER\_FLAG = False #Flag to activate 'start\_closing' variable, which measures the eyes closing time.

ALARM\_FLAG = False #Flag to check if alarm has ever been triggered.

#4. When the alarm is rang, count the number of times it is rang,and prevent the alarm from ringing continuously.

ALARM\_COUNT = 0 #Number of times the total alarm rang.

RUNNING\_TIME = 0 #Variable to prevent alarm going off continuously.

#5. . We should count the time eyes are being opened for data labeling.

PREV\_TERM = 0 #Variable to measure the time eyes were being opened until the alarm rang.

#6. Variables for trained data generation and calculation fps.

np.random.seed(9)

power, nomal, short = mtd.start(25)

test\_data = []

#The array the actual labeld data of test data is placed.

result\_data = []

#For calculate fps

prev\_time = 0

yawn\_counter=0

#7. Detect face & eyes.

print("loading facial landmark predictor...")

detector = dlib.get\_frontal\_face\_detector()

predictor = dlib.shape\_predictor("shape\_predictor\_68\_face\_landmarks.dat")

(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"]

#8. Run the cam.

print("starting video stream thread...")

vs = VideoStream(src=0).start()

time.sleep(1.0)

#9. Threads to run the functions in which determine the EAR\_THRESH.

th\_open = Thread(target = init\_open\_ear)

th\_open.deamon = True

th\_open.start()

th\_close = Thread(target = init\_close\_ear)

th\_close.deamon = True

th\_close.start()

while True:

frame = vs.read()

frame = imutils.resize(frame, width = 400)

rects = detector(frame,0)

#checking fps

prev\_time, fps = check\_fps(prev\_time)

for rect in rects:

shape = predictor(frame, 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)

#(leftEAR + rightEAR) / 2 => both\_ear.

both\_ear = (leftEAR + rightEAR) \* 500 #I multiplied by 1000 to enlarge the scope.

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)

if(yawn(shape[mStart:mEnd])>0.9):

cv2.putText(frame, "Yawn detected ", (10,130), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (200,30,20), 2)

if both\_ear < EAR\_THRESH :

if not TIMER\_FLAG:

start\_closing = timeit.default\_timer()

TIMER\_FLAG = True

COUNTER += 1

if COUNTER >= EAR\_CONSEC\_FRAMES:

mid\_closing = timeit.default\_timer()

closing\_time = round((mid\_closing-start\_closing),3)

if closing\_time >= RUNNING\_TIME:

if RUNNING\_TIME == 0 :

CUR\_TERM = timeit.default\_timer()

OPENED\_EYES\_TIME = round((CUR\_TERM - PREV\_TERM),3)

PREV\_TERM = CUR\_TERM

RUNNING\_TIME = 1.75

RUNNING\_TIME += 2

ALARM\_FLAG = True

ALARM\_COUNT += 1

print("{0}st ALARM".format(ALARM\_COUNT))

print("The time eyes is being opened before the alarm went off :", OPENED\_EYES\_TIME)

print("closing time :", closing\_time)

test\_data.append([OPENED\_EYES\_TIME, round(closing\_time\*10,3)])

result = mtd.run([OPENED\_EYES\_TIME, closing\_time\*10], power, nomal, short)

result\_data.append(result)

t = Thread(target = alarm.select\_alarm, args = (result, ))

t.deamon = True

t.start()

else :

COUNTER = 0

TIMER\_FLAG = False

RUNNING\_TIME = 0

if ALARM\_FLAG :

end\_closing = timeit.default\_timer()

closed\_eyes\_time.append(round((end\_closing-start\_closing),3))

print("The time eyes were being closed :", closed\_eyes\_time)

ALARM\_FLAG = False

cv2.putText(frame, "BOTH EAR : {:.2f}".format(both\_ear), (300,130), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (200,30,20), 2)

cv2.imshow("Frame",frame)

key = cv2.waitKey(1) & 0xFF

if key == ord("q"):

break

vs.release()

cv2.destroyAllWindows()

**7. CONCLUSION**

A non-intrusive system to localize the eyes and monitor fatigue was developed. Information about the face and eyes position are obtained through various algorithms. During the monitoring, the system is able to decide whether the eyes are opened or closed. When the eyes have been closed for few seconds, a warning signal is issued. In addition during monitoring, the system is able to automatically detect any eye localizing error that might have occurred.

**8. BIBLIOGRAPHY**

<https://www.analyticsvidhya.com/blog/2014/12/image-processing-python-basics/>

<https://www.geeksforgeeks.org/image-processing/>

<https://www.tutorialspoint.com/image-processing-in-python>

https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/#:~:targetText=The%20K%2Dnearest%20neighbors%20(KNN,performs%20quite%20complex%20classification%20tasks.