If the eye aspect ratio indicates that the eyes have been closed for a sufficiently long enough amount of time, we'll sound an alarm to wake up the driver:

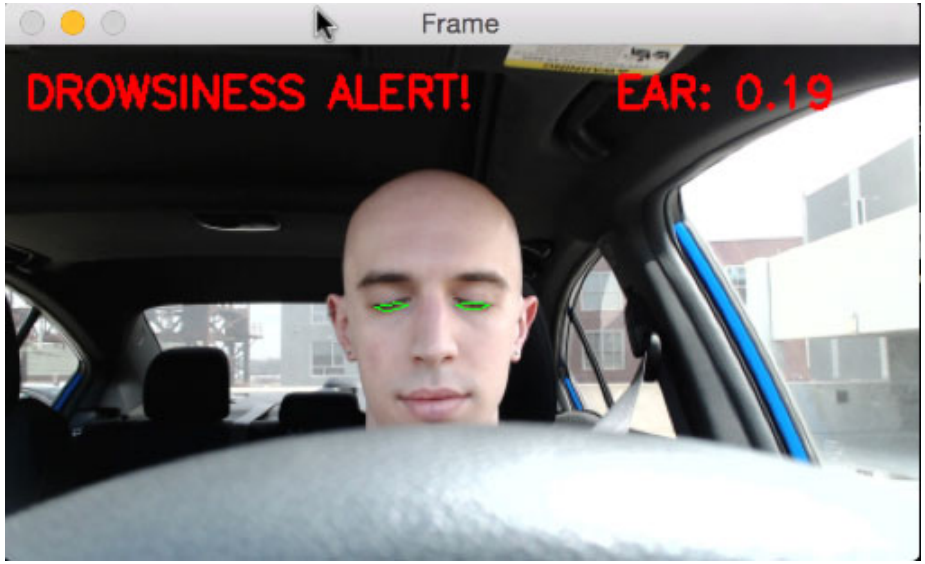

Figure 6: Step #4 — Sound an alarm if the eyes have been closed for a sufficiently long enough time.

In the next section, we'll implement the drowsiness detection algorithm detailed above using OpenCV, dlib, and Python.

Building the drowsiness detector with OpenCV

To start our implementation, open up a new file, name it `detect_drowsiness.py` , and insert the following code:

Drowsiness detection with OpenCV

```
1. # import the necessary packages
```

```
2. from scipy.spatial import distance as dist
3. from imutils.video import VideoStream
4. from imutils import face_utils
5. from threading import Thread
6. import numpy as np
7. import playsound
8. import argparse
9. import imutils
10. import time
11. import dlib
12. import cv2
```

Lines 2-12 import our required Python packages.

We'll need the **SciPy** package so we can compute the Euclidean distance between facial landmarks points in the eye aspect ratio calculation (not strictly a requirement, but you should have SciPy installed if you intend on doing any work in the computer vision, image processing, or machine learning space).

We'll also need the **imutils package**, my series of computer vision and image processing functions to make working with OpenCV easier.

If you don't already have `imutils` installed on your system, you can install/upgrade `imutils` via:

Drowsiness detection with OpenCV

```
1. $ pip install --upgrade imutils
```

We'll also import the `Thread` class so we can play our alarm in a separate thread from the main thread to ensure our script doesn't pause execution while the alarm sounds.

In order to actually play our WAV/MP3 alarm, we need the **playsound library**, a pure Python, cross-platform implementation for playing simple sounds.

The `playsound` library is conveniently installable via `pip`:

Drowsiness detection with OpenCV

```
$ pip install playsound
```

However, if you are using *macOS* (like I did for this project), **you'll also want to install pyobjc**, otherwise you'll get an error related to `AppKit` when you actually try to play the sound:

Drowsiness detection with OpenCV

```
$ pip install pyobjc
```

I *only* tested `playsound` on *macOS*, but according to both the documentation and Taylor Marks (the developer and maintainer of `playsound`), the library should work on Linux and Windows as well.

Note: *If you are having problems with* `playsound`*, **please consult their documentation** as I am **not** an expert on audio libraries.*

To detect and localize facial landmarks we'll need the **dlib library** which is imported on **Line 11**. If you need help installing dlib on your system, **please refer to this tutorial**.

Next, we need to define our `sound_alarm` function which accepts a `path` to an audio file residing on disk and then plays the file:

Drowsiness detection with OpenCV

```
14. def sound_alarm(path):
15.     # play an alarm sound
16.     playsound.playsound(path)
```

We also need to define the `eye_aspect_ratio` function which is used to compute the ratio of distances between the vertical eye landmarks and the distances between the horizontal eye landmarks:

Drowsiness detection with OpenCV

```
18. def eye_aspect_ratio(eye):
19.     # compute the euclidean distances between the two sets of
20.     # vertical eye landmarks (x, y)-coordinates
21.     A = dist.euclidean(eye[1], eye[5])
22.     B = dist.euclidean(eye[2], eye[4])
23.
24.     # compute the euclidean distance between the horizontal
25.     # eye landmark (x, y)-coordinates
26.     C = dist.euclidean(eye[0], eye[3])
27.
28.     # compute the eye aspect ratio
29.     ear = (A + B) / (2.0 * C)
30.
31.     # return the eye aspect ratio
32.     return ear
```

The return value of the eye aspect ratio will be approximately constant when the eye is open. The value will then rapid decrease towards zero during a blink.

If the eye is closed, the eye aspect ratio will again remain approximately constant, but will be *much smaller* than the ratio when the eye is open.

To visualize this, consider the following figure from Soukupová and Čech's 2016 paper, ***Real-Time Eye Blink Detection using Facial Landmarks***:

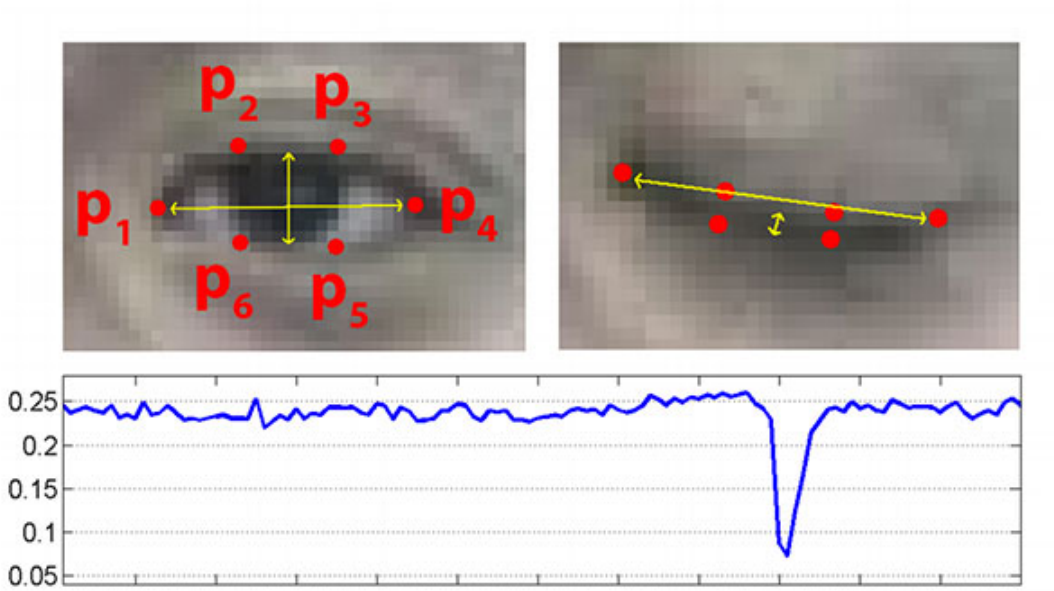

Figure 7: *Top-left:* A visualization of eye landmarks when then the eye is open. *Top-right:* Eye landmarks when the eye is closed. *Bottom:* Plotting the eye aspect ratio over time. The dip in the eye aspect ratio indicates a blink (Figure 1 of Soukupová and Čech).

On the *top-left* we have an eye that is fully open with the eye facial landmarks plotted. Then on the *top-right* we have an eye that is closed. The *bottom* then plots the eye aspect ratio over time.

As we can see, the eye aspect ratio is constant (indicating the eye is open), then rapidly drops to zero, then increases again, indicating a blink has taken place.

In our drowsiness detector case, we'll be monitoring the eye aspect ratio to see if the value *falls* but *does not increase again*, thus implying that the person has closed their eyes.

You can read more about blink detection and the eye aspect ratio in my **previous post**.

Next, let's parse our command line arguments:

Drowsiness detection with OpenCV

```
34. # construct the argument parse and parse the arguments
35. ap = argparse.ArgumentParser()
36. ap.add_argument("-p", "--shape-predictor", required=True,
37.    help="path to facial landmark predictor")
38. ap.add_argument("-a", "--alarm", type=str, default="",
39.    help="path alarm .WAV file")
40. ap.add_argument("-w", "--webcam", type=int, default=0,
41.    help="index of webcam on system")
42. args = vars(ap.parse_args())
```

Our drowsiness detector requires one command line argument followed by two optional ones, each of which is detailed below:

- `--shape-predictor` : This is the path to dlib's pre-trained facial landmark detector. You can download the detector along with the source code to this tutorial by using the ***"Downloads"*** section at the bottom of this blog post.
- `--alarm` : Here you can optionally specify the path to an input audio file to be used as an alarm.
- `--webcam` : This integer controls the index of your built-in webcam/USB camera.

Now that our command line arguments have been parsed, we need to define a few important variables:

Drowsiness detection with OpenCV

```
44. # define two constants, one for the eye aspect ratio to indicate
45. # blink and then a second constant for the number of consecutive
46. # frames the eye must be below the threshold for to set off the
47. # alarm
48. EYE_AR_THRESH = 0.3
49. EYE_AR_CONSEC_FRAMES = 48
50.
51. # initialize the frame counter as well as a boolean used to
52. # indicate if the alarm is going off
53. COUNTER = 0
54. ALARM_ON = False
```

Line 48 defines the `EYE_AR_THRESH`. If the eye aspect ratio falls *below* this threshold, we'll start counting the number of frames the person has closed their eyes for.

If the number of frames the person has closed their eyes in exceeds `EYE_AR_CONSEC_FRAMES` (**Line 49**), we'll sound an alarm.

Experimentally, I've found that an `EYE_AR_THRESH` of `0.3` works well in a variety of situations (although you may need to tune it yourself for your own applications).

I've also set the `EYE_AR_CONSEC_FRAMES` to be `48`, meaning that if a person has closed their eyes for 48 consecutive frames, we'll play the alarm sound.

You can make the drowsiness detector *more sensitive* by *decreasing* the `EYE_AR_CONSEC_FRAMES` — similarly, you can make the drowsiness detector *less*

sensitive by increasing it.

Line 53 defines `COUNTER`, the total number of *consecutive frames* where the eye aspect ratio is below `EYE_AR_THRESH`.

If `COUNTER` exceeds `EYE_AR_CONSEC_FRAMES`, then we'll update the boolean `ALARM_ON` (**Line 54**).

The dlib library ships with a **Histogram of Oriented Gradients-based** face detector along with a **facial landmark predictor** — we instantiate both of these in the following code block:

Drowsiness detection with OpenCV

```
56. # initialize dlib's face detector (HOG-based) and then create
57. # the facial landmark predictor
58. print("[INFO] loading facial landmark predictor...")
59. detector = dlib.get_frontal_face_detector()
60. predictor = dlib.shape_predictor(args["shape_predictor"])
```

The facial landmarks produced by dlib are an indexable list, as I describe **here**:

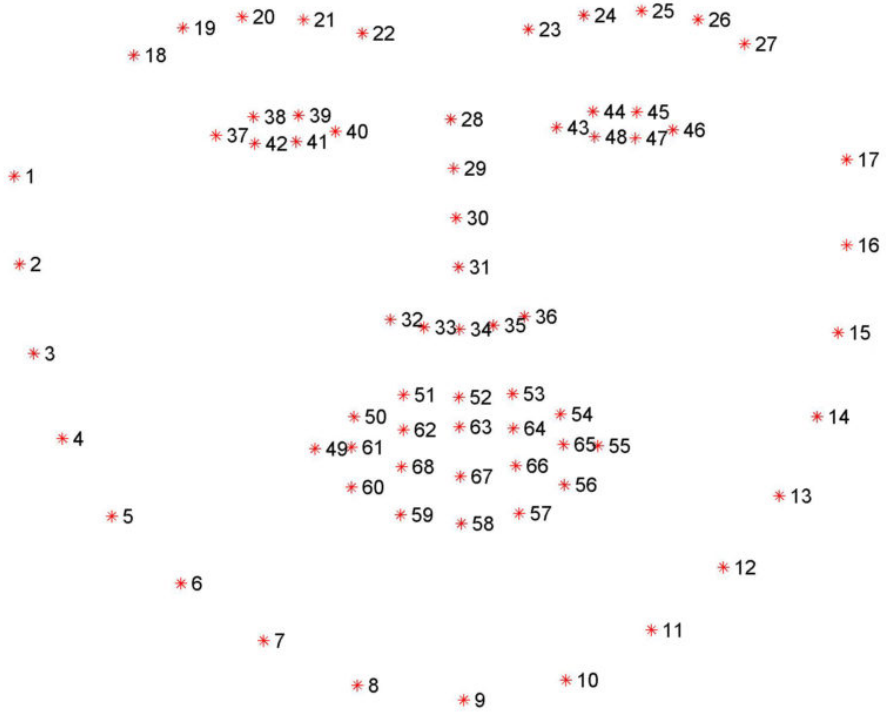

Figure 8: Visualizing the 68 facial landmark coordinates from the iBUG 300-W dataset (**larger resolution**).

Therefore, to extract the eye regions from a set of facial landmarks, we simply need to know the correct array slice indexes:

→ Click here to download the code

Drowsiness detection with OpenCV

```
62.  # grab the indexes of the facial landmarks for the left and
63.  # right eye, respectively
64.  (lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
65.  (rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
```

Using these indexes, we'll easily be able to extract the eye regions via an array slice.

We are now ready to start the core of our drowsiness detector:

Drowsiness detection with OpenCV

```
67. # start the video stream thread
68. print("[INFO] starting video stream thread...")
69. vs = VideoStream(src=args["webcam"]).start()
70. time.sleep(1.0)
71.
72. # loop over frames from the video stream
73. while True:
74.     # grab the frame from the threaded video file stream, resize
75.     # it, and convert it to grayscale
76.     # channels)
77.     frame = vs.read()
78.     frame = imutils.resize(frame, width=450)
79.     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
80.
81.     # detect faces in the grayscale frame
82.     rects = detector(gray, 0)
```

On **Line 69** we instantiate our `VideoStream` using the supplied `--webcam` index.

We then pause for a second to allow the camera sensor to warm up (**Line 70**).

On **Line 73** we start looping over frames in our video stream.

Line 77 reads the next `frame`, which we then preprocess by resizing it to have a width of 450 pixels and converting it to grayscale (**Lines 78 and 79**).

Line 82 applies dlib's face detector to find and locate the face(s) in the image.

The next step is to apply facial landmark detection to localize each of the important regions of the face:

→ Click here to download the code

Drowsiness detection with OpenCV

```
84.     # loop over the face detections
85.     for rect in rects:
86.         # determine the facial landmarks for the face region, then
87.         # convert the facial landmark (x, y)-coordinates to a NumPy
88.         # array
89.         shape = predictor(gray, rect)
90.         shape = face_utils.shape_to_np(shape)
91.
92.         # extract the left and right eye coordinates, then use the
93.         # coordinates to compute the eye aspect ratio for both eyes
94.         leftEye = shape[lStart:lEnd]
95.         rightEye = shape[rStart:rEnd]
96.         leftEAR = eye_aspect_ratio(leftEye)
97.         rightEAR = eye_aspect_ratio(rightEye)
98.
99.         # average the eye aspect ratio together for both eyes
100.        ear = (leftEAR + rightEAR) / 2.0
```

We loop over each of the detected faces on **Line 85** — in our implementation (specifically related to driver drowsiness), we assume there is only *one* face — the driver — but I left this `for` loop in here just in case you want to apply the technique to videos with *more than one* face.

For each of the detected faces, we apply dlib's facial landmark detector (**Line 89**) and convert the result to a NumPy array (**Line 90**).

Using NumPy array slicing we can extract the *(x, y)*-coordinates of the left and right eye, respectively (**Lines 94 and 95**).

Given the *(x, y)*-coordinates for both eyes, we then compute their eye aspect ratios on **Line 96 and 97**.

Soukupová and Čech recommend averaging both eye aspect ratios together to obtain a better estimation (**Line 100**).

We can then *visualize* each of the eye regions on our `frame` by using the `cv2.drawContours` function below — this is often helpful when we are trying to debug our script and want to ensure that the eyes are being correctly detected and localized:

Drowsiness detection with OpenCV

```
102.          # compute the convex hull for the left and right eye, then
103.          # visualize each of the eyes
104.          leftEyeHull = cv2.convexHull(leftEye)
105.          rightEyeHull = cv2.convexHull(rightEye)
106.          cv2drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
107.          cv2drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)
```

Finally, we are now ready to check to see if the person in our video stream is starting to show symptoms of drowsiness:

Drowsiness detection with OpenCV

```
109.          # check to see if the eye aspect ratio is below the blink
110.          # threshold, and if so, increment the blink frame counter
111.         if ear < EYE_AR_THRESH:
112.              COUNTER += 1
113.
114.          # if the eyes were closed for a sufficient number of
115.          # then sound the alarm
116.             if COUNTER >= EYE_AR_CONSEC_FRAMES:
117.              # if the alarm is not on, turn it on
118.                 if not ALARM_ON:
119.                      ALARM_ON = True
120.
121.                  # check to see if an alarm file was supplied,
122.                  # and if so, start a thread to have the alarm
123.                  # sound played in the background
```

```
124.                     if args["alarm"] != "":
125.                         t = Thread(target=sound_alarm,
126.                             args=(args["alarm"],))
127.                         t.deamon = True
128.                         t.start()
129.
130.             # draw an alarm on the frame
131.                 cv2.putText(frame, "DROWSINESS ALERT!", (10, 30),
132.                     cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
133.
134.         # otherwise, the eye aspect ratio is not below the blink
135.         # threshold, so reset the counter and alarm
136.         else:
137.             COUNTER = 0
138.             ALARM_ON = False
```

On **Line 111** we make a check to see if the eye aspect ratio is below the "blink/closed" eye threshold, `EYE_AR_THRESH` .

If it is, we increment `COUNTER` , the total number of *consecutive frames* where the person has had their eyes closed.

If `COUNTER` exceeds `EYE_AR_CONSEC_FRAMES` (**Line 116**), then we assume the person is starting to doze off.

Another check is made, this time on **Line 118 and 119** to see if the alarm is on — if it's not, we turn it on.

Lines 124-128 handle playing the alarm sound, provided an `--alarm` path was supplied when the script was executed. We take special care to create a *separate thread* responsible for calling `sound_alarm` to ensure that our main program isn't blocked until the sound finishes playing.

Lines 131 and 132 draw the text `DROWSINESS ALERT!` on our `frame` — again, this is often helpful for debugging, especially if you are not using the `playsound`

library.

Finally, **Lines 136-138** handle the case where the eye aspect ratio is *larger* than `EYE_AR_THRESH`, indicating the eyes are *open*. If the eyes are open, we reset `COUNTER` and ensure the alarm is off.

The final code block in our drowsiness detector handles displaying the output `frame` to our screen:

Drowsiness detection with OpenCV

```
140.            # draw the computed eye aspect ratio on the frame to help
141.            # with debugging and setting the correct eye aspect ratio
142.            # thresholds and frame counters
143.            cv2putText(frame, "EAR: {:.2f}".format(ear), (300, 30),
144.                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
145.
146.        # show the frame
147.        cv2.imshow("Frame", frame)
148.        key = cv2.waitKey(1) & 0xFF
149.
150.        # if the `q` key was pressed, break from the loop
151.        if key == ord("q"):
152.            break
153.
154.    # do a bit of cleanup
155.    cv2.destroyAllWindows()
156.    vs.stop()
```

To see our drowsiness detector in action, proceed to the next section.

Testing the OpenCV drowsiness detector

To start, make sure you use the ***"Downloads"*** section below to download the source code + dlib's pre-trained facial landmark predictor + example audio alarm file utilized in today's blog post.

I would then suggest testing the `detect_drowsiness.py` script on your local system in the comfort of your home/office *before* you start to wire up your car for driver drowsiness detection.

In my case, once I was sufficiently happy with my implementation, I moved my laptop + webcam out to my car (as detailed in the *"Rigging my car with a drowsiness detector"* section above), and then executed the following command:

Drowsiness detection with OpenCV

```
$ python detect_drowsiness.py \
    --shape-predictor shape_predictor_68_face_landmarks.dat \
    --alarm alarm.wav
```

I have recorded my entire drive session to share with you — you can find the results of the drowsiness detection implementation below: