# Drowsiness Detection

**Konstantinos Thanos, Mathematician - MSc**

## Main Idea

The main idea of this project is to detect drowsiness. In other words we are trying to detect eye blinks or yawns and decide about drowsiness. We will use **dlib** library for detecting faces and predicting the facial landmarks of a person.

```
import dlib
# DLIB - Face Detector
detector = dlib.get_frontal_face_detector()
# DLIB - Predictor
predictor = dlib.shape_predictor('Models/shape_predictor_68_
                                  face_landmarks.dat')
```

Using the above detector and predictor, after converting the frame (video capture) into grayscale, we detect faces on it. After that it's time to determine the facial landmarks.

```
while True:
    _, frame = cap.read()
    h, w = frame.shape[: 2]     # Height and Width of frame

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) # Grayscale

    faces = detector(gray, 0) # Detect faces in the gray frame

    # Loop through each face
    for face in faces:
        # Determine facial landmarks
        facial_landmarks = predictor(gray, face)
```

Below you can see a representation of the numbered facial landmarks.

Σχήμα 1: Facial Landmarks

## Eyes and Lips

Now that we have already detected the facial landmarks we can concentrate on the eyes and lips of a face in order to detect blinks or yawns respectively.

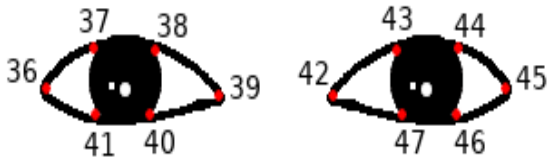
Before that we can define a function that will help us to calculate facial landmark point coordinates (x,y) for any part of a face, draw them on our frame (video capture) and return a numpy array with the corresponding points.

```
def draw_landmarks(face_part, landmarks):
    landmarks_list = []
    for point in face_part:
        x, y = landmarks.part(point).x, landmarks.part(point).y
        landmarks_list.append([x,y])
        cv2.circle(frame, (x,y), 2, (0,0,255), -1)
    return np.array(landmarks_list)
```

### Eyes

As shown in the facial landmarks image we can define two lists with numbers corresponding to left and right eye.

```
left_eye = [36,37,38,39,40,41]
right_eye = [42,43,44,45,46,47]
```




Σχήμα 2: Eye Landmarks

Using the function ***draw_landmarks*** we can save as numpy arrays the points (x,y coordinates) for left and right eye. Also we can draw the convex hull for each eye, as defined from the eye (left or right) landmark points.

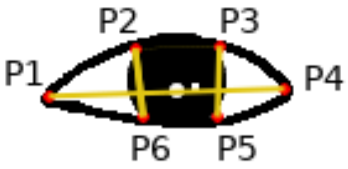
```
left_eye_points = draw_landmarks(left_eye, facial_landmarks)
right_eye_points = draw_landmarks(right_eye, facial_landmarks)

# Find and draw the convex hulls of left and right eye
left_eye_hull = cv2.convexHull(left_eye_points)
cv2.drawContours(frame, [left_eye_hull], -1, (0, 255, 0), 1)

right_eye_hull = cv2.convexHull(right_eye_points)
cv2.drawContours(frame, [right_eye_hull], -1, (0, 255, 0), 1)
```

Σχήμα 3: Eye Landmarks and Convex Hull

### Blink Detection

In order to build a function that will help us to determine if there is a blink or not we could use the **Eye Aspect Ratio** (E.A.R.) value as defined on the paper *Real-Time Eye Blink Detection using Facial Landmarks [Soukupova, Cech]*.

Let's enumerate the facial landmarks of a single eye with points $P_1$ to $P_6$ as shown below.




Σχήμα 4: Vertical and Horizontal distances

We will calculate the two vertical distances between points $P_2 - P_6$ (*dist1*) and $P_3 - P_5$ (*dist2*), and the horiontal (largest) distance between points $P_1 - P_4$ (*dist3*). So the E.A.R. (from now on **ear**) is defined as :

$$ear = \frac{(dist1 + dist2)}{2 \cdot dist3}$$

```
# Function to calculate eye aspect ratio
def eye_aspect_ratio(eye):
    # Vertical distances
    dist1 = dist.euclidean(eye[1], eye[5]) # P2–P6
    dist2 = dist.euclidean(eye[2], eye[4]) # P3–P5
    # Horiontal distance
    dist3 = dist.euclidean(eye[0], eye[3]) # P1–P4

    # Eye Aspect Ratio (E.A.R.)
    ear = (dist1 + dist2) / (2.0 * dist3)

    return ear
```

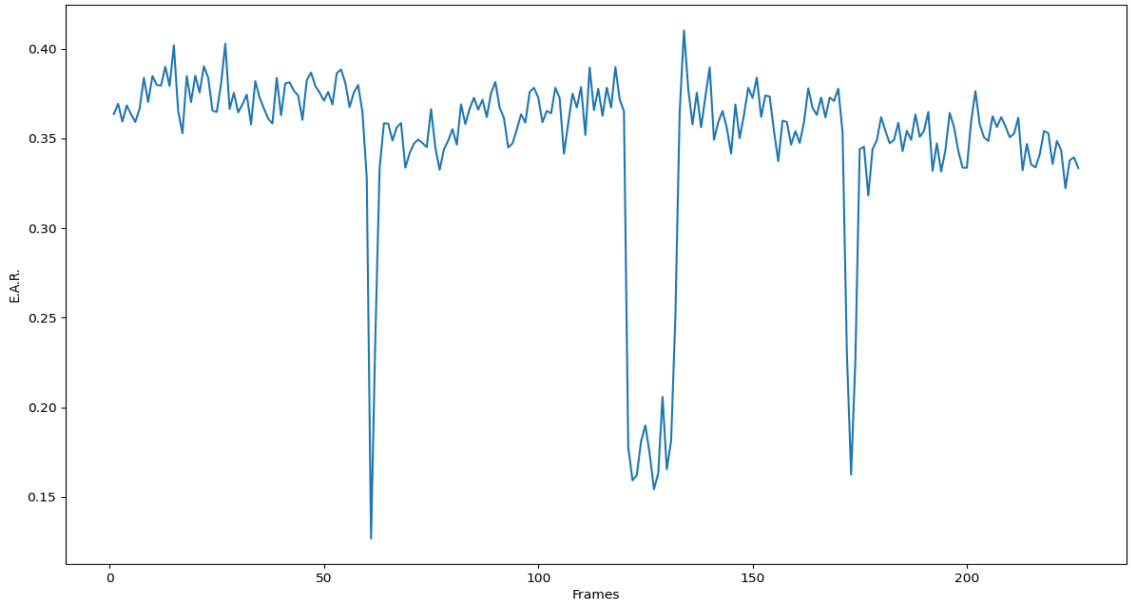
The above explanation and calculation is referring to only one eye. In order to use both eyes' *ear*, we can find the average eye aspect ratio from the two eyes (left and right).

```
left_ear = eye_aspect_ratio(left_eye_points)     # Left eye aspect ratio
right_ear =  eye_aspect_ratio(right_eye_points) # Right eye aspect ratio
ear = (left_ear + right_ear) / 2.0       # Average eye aspect ratio
```

### Is it blink or not ?

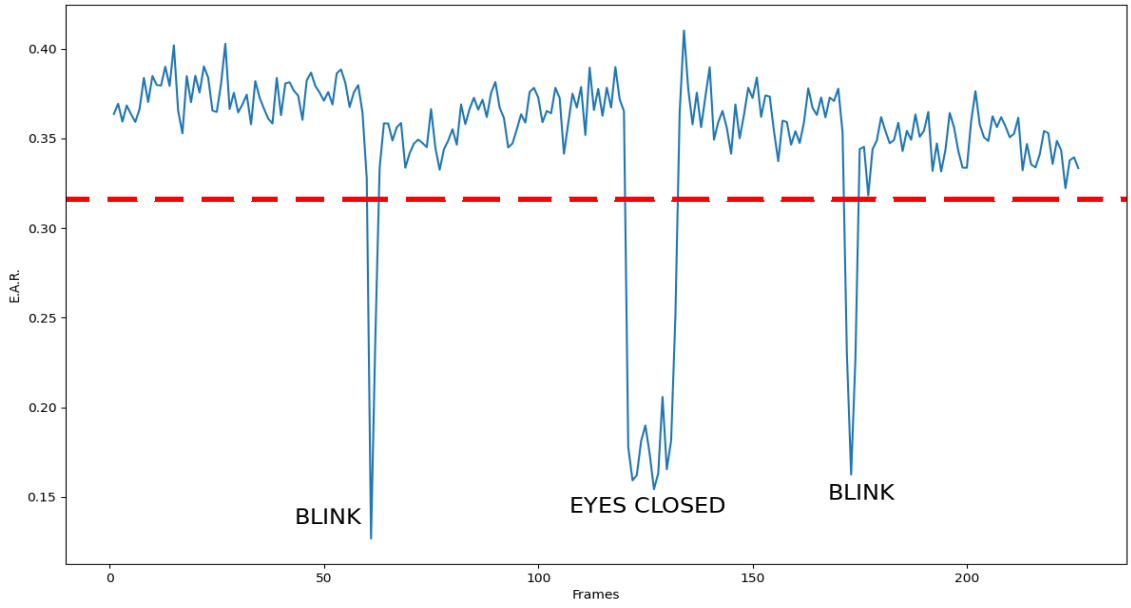
How to decide if there is a blink or not by the *ear* value ? We can set an *ear* threshold. This means that if the current value is less than the threshold then it is a blink.

How to do this ? We can "print" a plot showing the *ear* value for each frame. Below there is a simple example that I made to define my *ear* threshold value.




Σχήμα 5: *E.A.R.* values per frame

In this plot ( 250 frames) we can see that most of the *ear* values are less than 0.30 and there are 3 times that we see that difference. These 3 times correspond to 2 instant blinks and 1 time with eyes closed.




Σχήμα 6: Blinks and eyes closed intervals

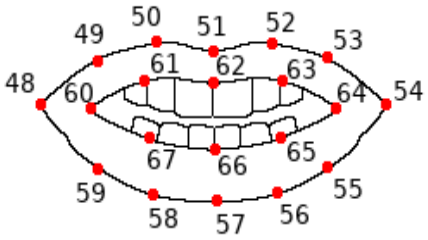
As a result I will set my *ear* threshold value to 0.3 at the beggining of our code.

```
ear_thresh = 0.3 # E.A.R. threshold
```

## Lips

To find out if there is a yawn or not we will work exactly as before on the eyes section. First let's define a list with numbers corresponding to the lips part of a face as shown in the facial landmarks image.

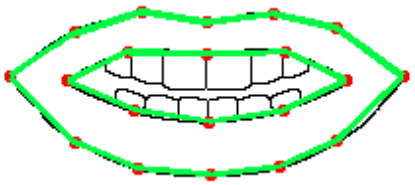
```
lips = [60,61,62,63,64,65,66,67] # Landmark indexes for lips
```




Σχήμα 7: Lips landmarks

We will use as before the ***draw_landmarks*** function to save as numpy array the lips facial landmark point coordinates (x,y) and draw them on frame. We will also calculate and draw the convex hull that is define by them.

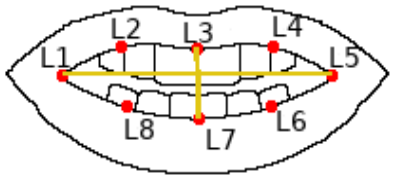
```
lips_points = draw_landmarks(lips, facial_landmarks)
lips_hull = cv2.convexHull(lips_points)
cv2.drawContours(frame, [lips_hull], -1, (0, 255, 0), 1)
```



Σχήμα 8: Lips landmarks and Convex Hull

### Yawn Detection

We will define a function similar to ***eye_aspect_ratio*** function, in order to calculate **Lips Aspect Ratio** (L.A.R.). Let's enumerate the facial landmarks corresponding to the "interior" lips with points $L_1$ to $L_8$.




Σχήμα 9: Vertical and Horiontal distance

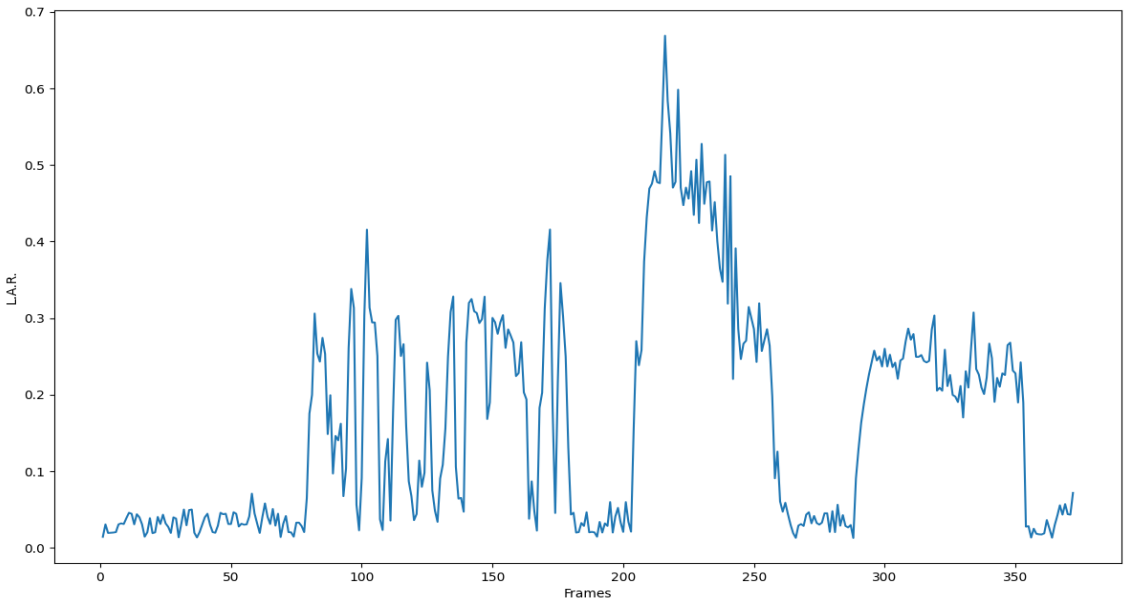
```python
# Function to calculate lips aspect ratio in the same way as in E.A.R.
def lips_aspect_ratio(lips):
    # Vertical distance
    dist1 = dist.euclidean(lips[2], lips[6]) # L3–L7
    # Horiontal distance
    dist2 = dist.euclidean(lips[0], lips[4]) # L1–L5

    # Lips Aspect Ratio (L.A.R.)
    lar = float(dist1/dist2)

    return lar
```

Now we can calculate the vertical distance between points $L_3$ and $L_7$ (*dist1*) and the horiontal distance between points $L_1$ and $L_5$ (*dist2*). So the L.A.R. (from now on **lar**) is defined as :

$$lar = \frac{dist1}{dist2}$$

### Is it yawn or not ?

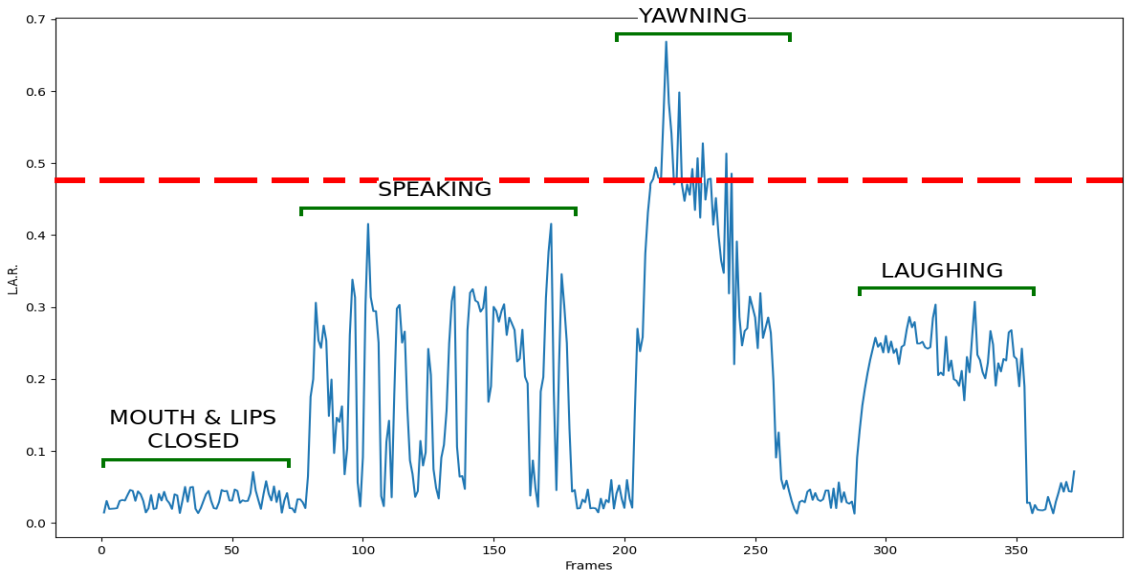
As in blink detection, we are going to set a *lar* threshold. Here it means that if the current value is greater than the threshold then it is a yawn.




Σχήμα 10: *L.A.R.* values per frame

This plot is way different that the one on *ear* threshold values per frame. Here there are 4 different time periods (frame periods) with different situations. These situations are :

1. Lips closed
2. Speaking
3. Yawning
4. Laughing




Σχήμα 11: Lips closed, speaking, yawning and laughing intervals

As a result I will set my *lar* threshold value to 0.5 at the beggining of our code.

```
lar_thresh = 0.5 # L.A.R. threshold
```

## Drowsiness detection

By combining eye aspect ration (*ear*) and lips aspect ratio *lar*) values we can deside if there is drowsiness or not.

```
if total_yawns > 2 or total_blinks > 3:
    cv2.putText(frame, "ALERT", (w-120, 160), font, 1.2, (0, 0, 255), 4)
```

## References


1. Real-Time Eye Blink Detection Using Facial Landmarks [Soukupová, Čech]
2. Eye blink detection with OpenCV, Python, and dlib [Adrian Rosebrock - pyimagesearch]