Driver Drowsiness Detection

Katrin Glöwing

May 2020

Contents

1 Introduction

Three seconds of sleep at a speed of 100 km/h are enough to cover 80 meters blind and thus without control. Human lives are at risk by serious accidents through this short time of losing control. Out of 1000 car drivers, one in four, according to a survey by the *German Road Safety Council (DVR)*, has nodded at least once while driving [21].
The diagram (figure 1) below shows the accidents with personal injury due car driver fatigue in the ferderal territory. The data are from the *Federal Office of Statistics (Destatis)*. On the y-axis is the number of personal injury accidents (0 - 250) and on the x-axis is the time from February 2019 to February 2020. The minimum is 110 in January this year (2020) but this bar chart only shows the personal injury accidents and not how many people are affected. In January 2020 were one casualty, 58 seriously and 107 slightly injured. In July 2019 the number of personal injury accidents reach the peak at 233 accidents, which means in this case: five casualty, 125 seriously and 287 slightly injured [3].
For these reasons, drowsiness detection of the driver is important.

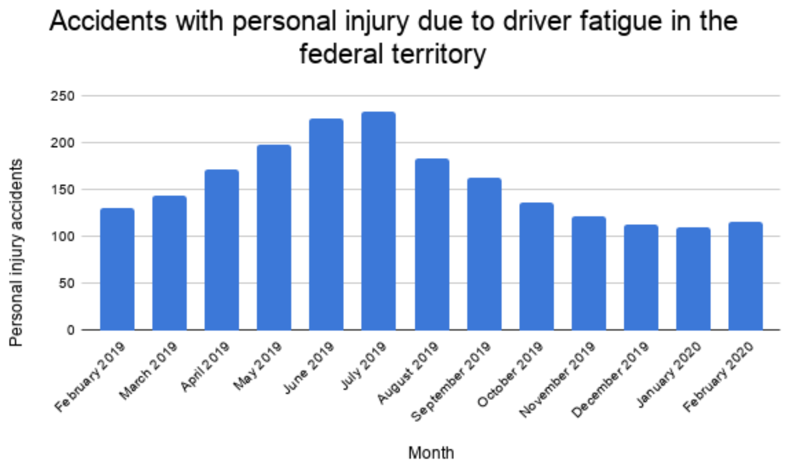


Figure 1: Statistics: Accidents due to driver fatigue [3]

In the next section will explain some related works of which most are already used. The following section is the concept. There will specify the goal of this project, the requirements and the use cases. The Implementation of the project will be shown in the next section. Descriptions of face detection, face landmark detection, specific facial structures detection, real-time eye blink detection and the drowsiness detection system are included. The drowsiness detection system is subdivided in the explanation of necessary packages, alarm sound function and eye aspect ratio function, command prompt arguments, initialization and prediction of facial landmark detection and the main part of the program. After that there is a evaluation of this project, in this is a program showcase and the implemented requirements invited. Finally in this composition is a summary.

2 Related Works

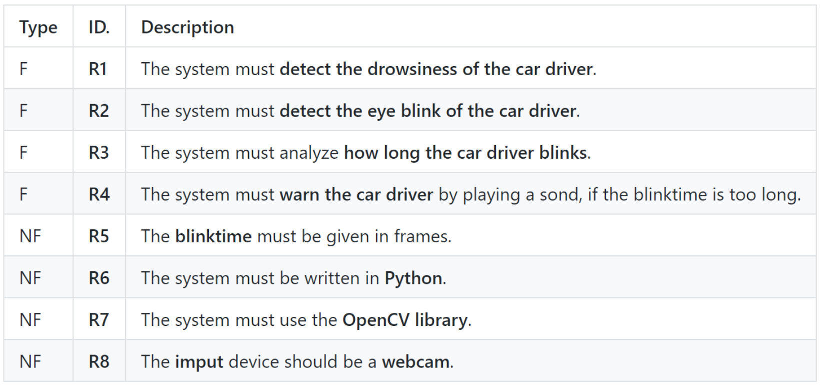
For these reasons mentioned in section *1 Introduction*, to reduce traffic accidents on road, it is beneficial to recognize fatigue and warn the driver.
Drowsiness can for example be detected by using heart rate variability [22] or by analyzing the vehicle steering (steering angle) [19] or by determine the activity of the head, eyes respectively the eyelids [7].
The detection on the fatigue of the driver is already on the market. There is for example the *Bosch Driver Drowsiness Detection* [13], the *Driver Alert System* from *VW* [23] which is also used in some *Škoda* models [1] and the *ATTENTION ASSIST-System* from *Daimler* which is fitted as standard in two *Mercedes-Benz* Classes [2]. All this examples are identify the drowsiness of the driver by analyzing the vehicle steering.
There is also a approach from *VW* which uses precision measurement of eyelid movements for determining the drivers tiredness level [7].

3 Concept

3.1 Goal of the project

The goal of my project is to implement a system, which detects the drowsiness of the car driver by determine the activity of the eyes respectively the eyelids.

3.2 Requirements



Type	ID.	Description
F	**R1**	The system must **detect the drowsiness of the car driver**.
F	**R2**	The system must **detect the eye blink of the car driver**.
F	**R3**	The system must analyze **how long the car driver blinks**.
F	**R4**	The system must **warn the car driver** by playing a sond, if the blinktime is too long.
NF	**R5**	The **blinktime** must be given in frames.
NF	**R6**	The system must be written in **Python**.
NF	**R7**	The system must use the **OpenCV library**.
NF	**R8**	The **imput** device should be a **webcam**.

Figure 2: Requirements of the project

Figure 2 presents the requirements of the project. In summary they are eight requirements. Four of them are functional (Type: F) and four are non-functional (Type: NF). Functional requirements are requirements which describes what a system or product must execute [15] and non-functional requirements describes the fundamental basics of the systems architecture [4].
The system must detect the drowsiness of the car driver (ID.: R1) by detecting the eye blink of the car driver (ID.: R2). It is also important to analyze how long the car driver blinks (ID.: R3) and if the car driver blinks too long, which means he/she falls asleep, the system must play a sound to warn the car driver (ID.: R4).
The blinktime, which must be given in frames (ID.: R5), Python as programming language (ID.: R6), using the OpenCV library (ID.: R7) and the webcam as an imput device (ID.: R8) are the non-functional requirements of ths project.

3.3 Use Cases

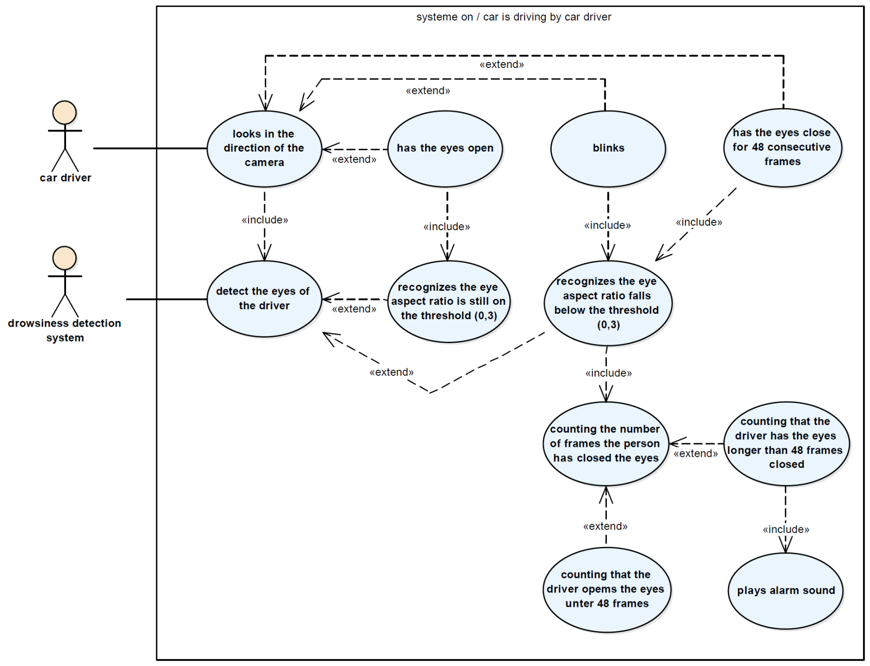

systeme on / car is driving by car driver
car driver
drowsiness detection system
looks in the direction of the camera
has the eyes open
blinks
has the eyes close for 48 consecutive frames
detect the eyes of the driver
recognizes the eye aspect ratio is still on the threshold (0,3)
recognizes the eye aspect ratio falls below the threshold (0,3)
counting the number of frames the person has closed the eyes
counting that the driver has the eyes longer than 48 frames closed
counting that the driver opems the eyes unter 48 frames
plays alarm sound
«extend»
«include»

Figure 3: UML: Use Case Model.

Figure 3 shows the Use Cases of this Project. There are two actors, the *car driver* and the *drowsiness detection system* itself. How the actors act when the *system is on* respectively when the *car is driven by the car driver* is presented in the boundary. First, the car driver *looks in the direction of the camera.* The driver drowsiness system, *detects the eyes of the driver.* The car driver can now *have the eyes open*, *blink* or *close their eyes for 48 consecutive frames*, which means the driver falls asleep. This is presented by *extend* relations. If the car driver *has the eyes open*, the system will *recognize the eye aspect ratio is still on the threshold* which is 0,3 and signify that the eyes are completely open. If the car driver *blinks* or *has their eyes close for 48 consecutive frames*, the system will *recognize the eye aspect ratio falls below the threshold.* After that, the system is *counting the number of frames that the driver has closed the eyes.* This use case can be extend by two options. The first option is, that the system is *counting that the driver has their eyes longer than 48 frames closed.* If that is the case, the system will *play an alarm sound* to warn respectively to wake the driver up. The second option is that the system is *counting that the driver opens the eyes under frames*, which means the driver only blinked.

4 Implementation

4.1 Face Detection

The first and a crucial step for detecting the drowsiness of the car driver by determine the activity of the eyes respectively the eyelids is to identify and align the face. This is useful for human visual perception [12].
Since February 2014, the object detection technique *Histograms of Oriented Gradients for Human Detection (HOG)* is in the dlib library. The HOG technique is from Navneet Dalal and Bill Triggs and is a feature descriptor for object detection [5].

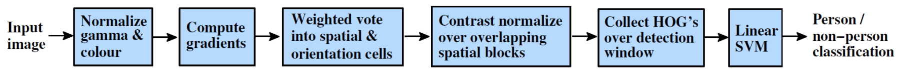

Input image
Normalize gamma & colour
Compute gradients
Weighted vote into spatial & orientation cells
Contrast normalize over overlapping spatial blocks
Collect HOG's over detection window
Linear SVM
Person / non−person classification

Figure 4: HOG's feature extraction and object detection chain [5].

Figure 4 shows an overview of HOG's feature extraction and object detection chain. The *input* is an *image* and the first step is the *normalization of gamma and colour* which offers the possibility for colour-based object recognition techniques. The next step is to *compute gradients* [5].

Gradient values are mapped from 0 to 255, 0 is white and 255 is black [14].
As in Figure 5 is shown, they are two different gradients: the horizontal gradient (Figure 5: second picture) and the vertical gradient (Figure 5: third picture). That means, if the first picture in Figure 5, the original picture, will calculate gradients in a horizontal direction, from left to right, the second picture will appear. If the gradient will be calculated in a vertical direction, from top to bottom, the outcome is the third picture. Horizontal and vertical gradients are being calculated by subtracting pixels [14].
If pixels have large positive changes in the original picture, the pixels will be

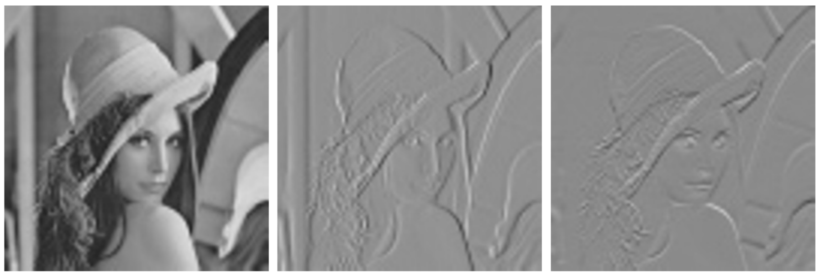


Figure 5: Calculate the amount of the gradient [14].

white in the gradient picture. Pixels with large negative changes will be black and pixels with less or no changes will be gray [14].
An example: In the original picture the hat is because of the lights on the left side much brighter than the background. That means in this area the background has much higher values than the hat. By calculating the gradients in horizontal direction, this is essential because the transition from higher to lower values is given and the transition has large positive changes which leads to white pixels in this area in the gradient picture. But if the gradients are calculated in a vertical direction, shown in the third picture, this specific area on the left side of the head is grey because when the calculation is in a vertical direction, they are less or no changes in pixels [14].
If the horizontal $s_x(x, y)$ and the vertical $s_y(x, y)$ gradient is calculated, the values of the two computed gradients will build a feature vector. The next step will be to calculate the gradient magnitude $g_b(x, y)$ [14]:

$$g_m\left(x, y\right) = \sqrt{s_x\left(x, y\right)^2 + s_y\left(x, y\right)^2} \tag{1}$$

Now it is possible to evaluate the gradient angle, which describes the orientation of the gradient:

$$g_a = arctan\left(\frac{s_y(x, y)}{s_x(x, y)}\right) \tag{2}$$

After that the gradient is a oriented.
The next step in the HOG's feature extraction and object detection chain (Figure 4) is *weighted vote into spatial & orientation cells* which is the decisive non-linearity of the descriptor. The orientation of every pixel centred gradient element is the base for the calculation of a weighted vote for an edge orientation histogram channel. The votes are concentrated in cells, which are orientation bins over local spatial [5].
After that, the next step is to *normalize the contrast over overlapping spatial blocks* (Figure 4). For good performance is a effective normalization of local contrast necessary. Because of local variability in lighting and of the foreground-background contrast the gradient strength diversify over a wide range. The effective normalization of local contrast settles this. Normalization schemes are

based on classifying the cells in spatial blocks. Each of this blocks will be contrast normalized separately. These blocks will be overlap. The final descriptor vector is then a unification of various components of each scalar cell response. The normalisation of each scalar cell response will be execute with respect to a different block [5].
The next step is *collecting HOG's over detection window* (Figure 4). The size of the detection window is 64x128 pixel. Around the object or person are 16 pixels of border, which are contained in the detection window. The detection will be easier thought this relevant amount of context [5].
The last step is the *linear Support Vector Machine (SVM)* which is the classifier [5]. In year 2005 the Figure 4 has been published in the article from N. Dalal and B. Triggs. Today the HOG technique uses the Max-Margin Object Detection (MMOD) [10].
The recognition of objects in pictures is learned via MMOD. MMOD optimize across all sub-windows and does not sub-sample like other object detection methods. This leads to significant increases in performance. Any object detection method which is linear in the learned parameters, like HOG, can be enhance by MMOD [11].
The output of *HOG's feature extraction and object detection chain* (Figure 4) is the *classification of a person or the classification that no person is in the detection window* [5].

4.2 Facial Landmark Detection

If the detection of the face is successfully, the next step will be to align the face. The system uses the facial landmark detection which is implemented inside dlib and pre-trained.
The algorithm guess the exact position of facial landmarks accurately in a computer-based efficient way by employing a cascade of regressors [8].

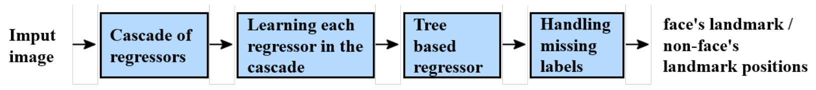

Imput image
Cascade of regressors
Learning each regressor in the cascade
Tree based regressor
Handling missing labels
face's landmark / non-face's landmark positions

Figure 6: Individual components of the cascade and this method perform training [12].

The first step of this algorithm is, as you can see in Figure 7, *the cascade of regressors*. An update vector will be predicted from the image and the current from each regressor in the cascade. Next, to improve the estimate, this vector will be add to the current shape estimate [8].
The second step is *learning each regressor in the cascade* (Figure 7). If in the training data has for each face image also its shape vector, it is possible to create an initial shape estimate and the target update step from the training data triples of a face image. Now the algorithm learn the first regression function.

Constant sampling from the trained shape vectors each initial shape estimate for an image without replacement is the following step. By utilize gradient tree boosting with the sum of square error loss the algorithm learn the regression function. Via resetting the data, for the next regression, the set of training triples is updated supply the training data. If the cascade from a certain number of regressors is learned which in combination result give a adequate level of accuracy, the iteration of this process will stop. In this step of the algorithm (Figure 7: *learning each regressor in the cascade*), the tree based regressors were also verified by reviewing the most important implementation details [8].
The last step of this in Figure 7 shown algorithm chain is *handling missing labels*. This is detached with weight factors. By adding new variables for each training image and landmark, the algorithm shows per setting a 0, represents that the landmark is not labeled in this image. If the algorithm set an 1, the landmark will be label in this image [8].
Figure 7 shows that the output will be the face's landmark positions or no face's landmarks positions, if there is no face on the image [8].

4.3 Specific Facial Structures Detection

Via training the facial landmark detector, which is mention in section 4.2, on the *iBUG 300-W dataset*, the facial landmark detector creates a 68 point mapping (68 x-, y-coordinates) that relates to specific facial structures [9]. The dataset is a semi-automatic annotation methodology for commenting solid face datasets [18].

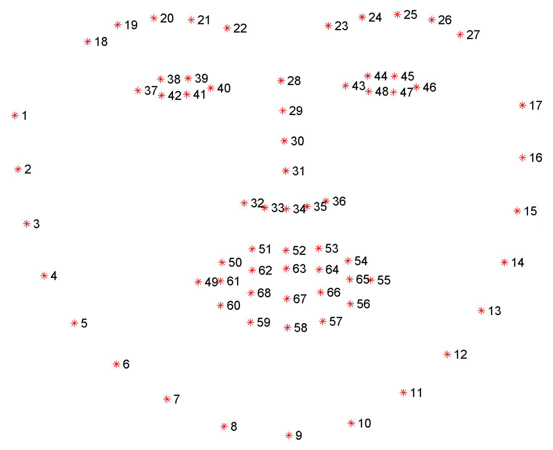


Figure 7: The by iBUG 300-W dataset used 68 points mark-up [18].

Figure 7 represents the location of the 68 (x, y)-coordinates [18]. These facial regions can be accessed via Python indexing. This project is about detecting the drivers drowsiness, which means the focus is on the eyes. The right eye is

specified through the indexing 36 to 42 and the left eye is specified thought 42 to 48 [16]. The indexes begin one point earlier than it is shown in Figure 7 because of the Python indexing that is ”zero indexed”. This means, the algorithm start the at zero and when the 36th item in the list is searched, than it will be the landmark 37 because the landmarks starts with one [6].
In the *face_utils* of the *imutils library* are this classifications of mouth, right eyebrow, left eyebrow, right eye, left eye, nose and jaw already encoded. Removing the indexes into facial landmarks arrays and supplying a string as a key to extract various facial features is possible with this dictionary [16].

4.4 Real-Time Eye Blink Detection

Interesting for this project are the eyes. Every eye is classified via 6 (x, y)-coordinates (refer to Figure 7). The coordinates numbers start on the left site (view: looking at this person) and count up clockwise around each eye [17].

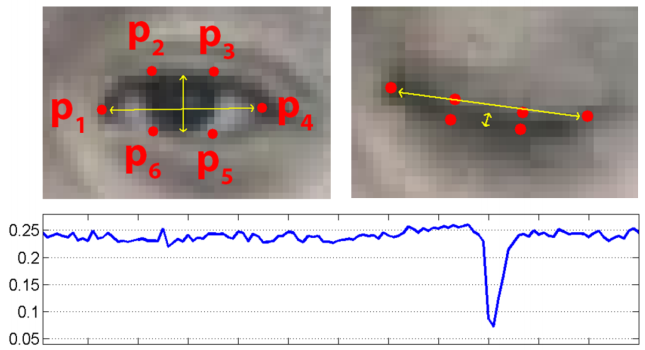


Figure 8: top left: open eye with landmarks, top right: closed eye with landmarks, bottom: example eye aspect ratio plot [20].

Figure 8 (top left) shows this order of points on a real eye. There are also two arrows which represents the width and the height of these coordinates. The relation between this length information can be calculated by the following formula by using the facial landmarks (p_1, ..., p_6). The result of the equation is the eye aspect ratio (EAR) [20].

$$ERA = \frac{||p_2 - p_6|| + ||p_3 - p_5||}{2 \times ||p_1 - p_4||} \tag{3}$$

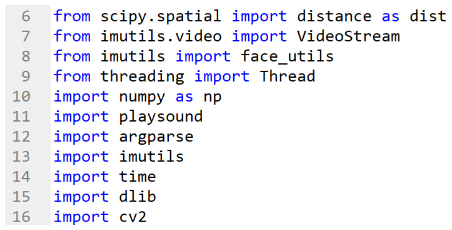
The formula represents the space from the height divided by two times of the distance from the width. The distance from the width is twice because the distance from the height has one more pair of facial landmarks [20].
Eye blink detection is possible with this formula. If the eye is open (compare Figure 8 top left), the ERA will keep stable. But if the driver blinks (compare

Figure 8 top right), the ERA will decrease rapidly to almost zero. The line chart in Figure 8 on the bottom evidences the EAR over a time for a video clip. The graph keeps stable for many frames on a EAR value about 2.25 EAR, than the EAR makes a reduce fall over a half frame to a value close to zero but then made a significant recovery in the next half frame again to a value about 2.25 ERA. This means transferring, the car driver blinks [20].

4.5 Drowsiness Detection System

The implementation of this project is written in *Python* and uses the OpenCV library [17].

4.5.1 Necessary Packages



```python
from scipy.spatial import distance as dist
from imutils.video import VideoStream
from imutils import face_utils
from threading import Thread
import numpy as np
import playsound
import argparse
import imutils
import time
import dlib
import cv2
```

Figure 9: Importing the necessary packages [17].

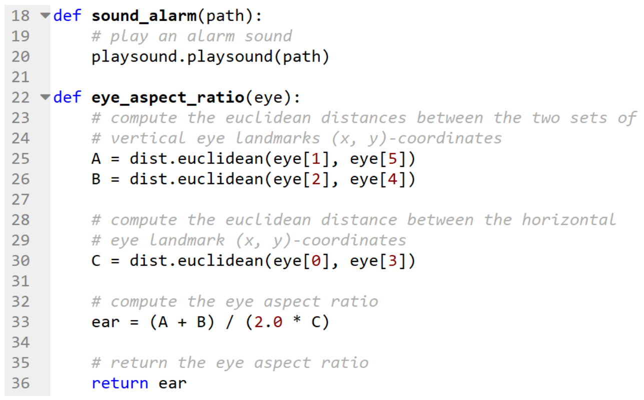
Figure 9 lists all necessary packages that are imported in this project:

- The *SciPy package* is useful for calculation the eye aspect ratio with the help of facial landmarks [17].
- The *imutils package* makes working with OpenCV easier in case of working with computer vision and image processing functions [17].
- The *thread class* offers playing an alarm sound in a separate thread from the main thread. This prevents that the system stops while the alarm sound is on [17].
- The *playsound library* arranges for playing a simple WAV/MP3 alarm [17].
- The *dlib library* is needed to detect and localize facial landmarks [17].

4.5.2 Alarm Sound Function and EAR Function

Figure 10 presents the two functions of this program.

- The *sound_alarm function* is defined to play the file after supposing a path (compare section 4.5.1 *thred class*) to the audio file [17].



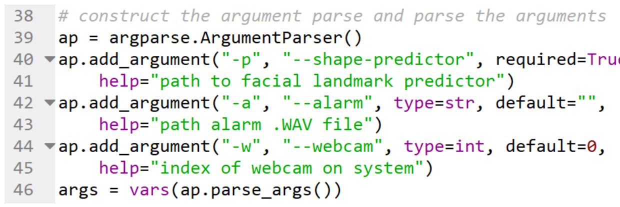
```python
18 def sound_alarm(path):
19     # play an alarm sound
20     playsound.playsound(path)
21
22 def eye_aspect_ratio(eye):
23     # compute the euclidean distances between the two sets of
24     # vertical eye landmarks (x, y)-coordinates
25     A = dist.euclidean(eye[1], eye[5])
26     B = dist.euclidean(eye[2], eye[4])
27
28     # compute the euclidean distance between the horizontal
29     # eye landmark (x, y)-coordinates
30     C = dist.euclidean(eye[0], eye[3])
31
32     # compute the eye aspect ratio
33     ear = (A + B) / (2.0 * C)
34
35     # return the eye aspect ratio
36     return ear
```

Figure 10: Sound alarm function and eye aspect ratio function [17].

- The *eye_aspect_ratio function* is apply to calculate the EAR, which is illustrated in 4.4. A and B are the distances of the height and C is the distance of the width. Referring to section 4.3, the items of A are one and five and the items of B are two and four because the Python indexing is "zero indexed". In Line 33 (Figure 10) is the formula which is also already explained in section 4.4. Line 36 *returns the ear*, the result of the equation [17].

4.5.3 Command Prompt Arguments

Figure 11 shows the connection to the command prompt enabled by the *argparse package* that is displayed in Figure 9 [17].

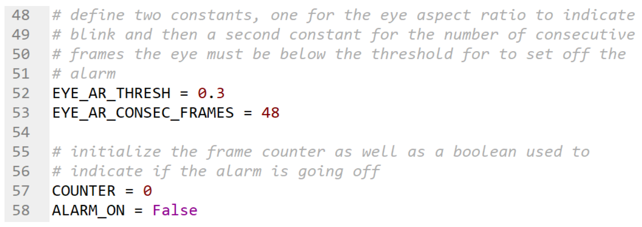


```python
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--shape-predictor", required=True,
    help="path to facial landmark predictor")
ap.add_argument("-a", "--alarm", type=str, default="",
    help="path alarm .WAV file")
ap.add_argument("-w",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

Figure 11: Constructing and parsing the argument parse arguments [17].

- For getting to the path of the pre-trained face marking detector is the *--shape-predictor command line argument*. This argument has been filled, the other to arguments are optional ones [17].
- To define the path to an input audio file is possible with the *--alarm argument*. This is for choosing the alarm sound [17].
- The *--webcam argument* is for choosing the webcam. The data type is an integer [17].

4.5.4 Definition of the Variables

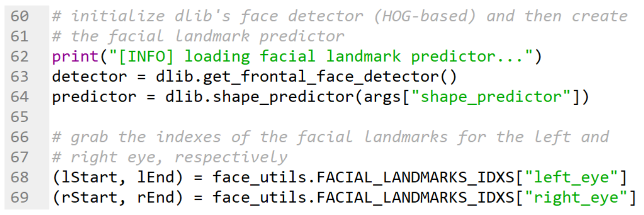


```
48 # define two constants, one for the eye aspect ratio to indicate
49 # blink and then a second constant for the number of consecutive
50 # frames the eye must be below the threshold for to set off the
51 # alarm
52 EYE_AR_THRESH = 0.3
53 EYE_AR_CONSEC_FRAMES = 48
54
55 # initialize the frame counter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

Figure 12: Defining a few important variables [17].

Line 52 in Figure 12 defines the *EYE_AR_THRESH* as a threshold from 0.3. The maximum number of frames that the eyes could be closed without playing the alarm sound is defined in line 53 (*EYE_AR_CONSEC_FRAMES*) as 48 frames. The *COUNTER* is defined in line 57 and count the number of consecutive frames where the eyes are closed. At the beginning the alarm sound is off (*ALARM_ON*, line 58) [17].

4.5.5 Initalization and Predicition of the Facial Landmark Detection



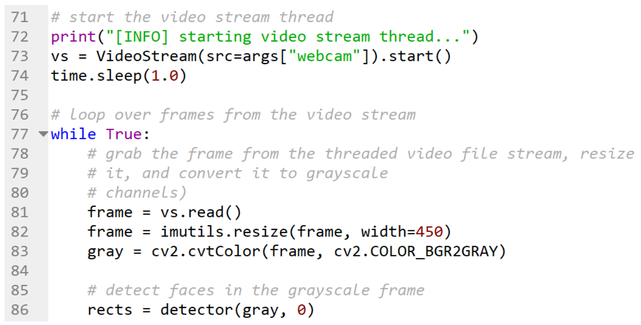
```
60 # initialize dlib's face detector (HOG-based) and then create
61 # the facial landmark predictor
62 print("[INFO] loading facial landmark predictor...")
63 detector = dlib.get_frontal_face_detector()
64 predictor = dlib.shape_predictor(args["shape_predictor"])
65
66 # grab the indexes of the facial landmarks for the left and
67 # right eye, respectively
68 (lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
69 (rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
```

Figure 13: Initialize the face detector and then create the facial landmark predictor and grabbing the indexes of the eyes [17].

In Figure 13 the dlib's face detector that is commented in section 4.1 is initialized (line 63) and creates the facial landmark predictor (line 64) which is commented in section 4.2. Now for extracting the eye regions, the usage of the correct array slice indexes is needed (line 68 and line 69) [17].

4.5.6 Main Part of the Program

In Line 73 in Figure 14 the *VideoStream* will be initiate. After that the program takes a short brake for the camera sensor to open the camera. The loop over frames from the video stream is started in line 77. After reading the next frame in line 81, this frame will be resized (line 82) and will be converted to gray scale (line 83). To find and locate the face(s), the Dlib's face detector is inserted in line 86 [17].



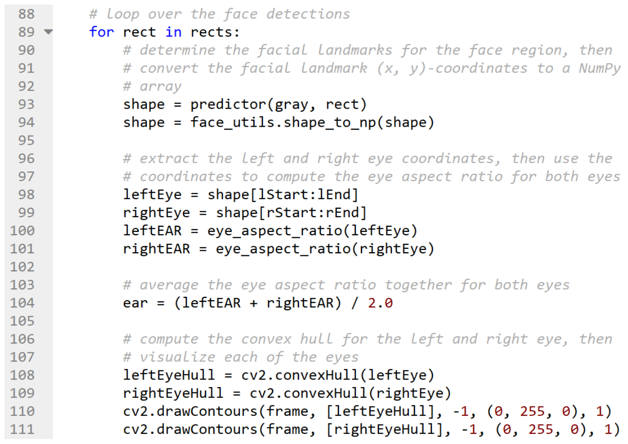
```python
# start the video stream thread
print("[INFO] starting video stream thread...")
vs = VideoStream(src=args["webcam"]).start()
time.sleep(1.0)

# 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)
    frame = vs.read()
    frame = imutils.resize(frame, width=450)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # detect faces in the grayscale frame
    rects = detector(gray, 0)
```

Figure 14: Initialize the face detector and then create the facial landmark predictor and grabbing the indexes of the eyes [17].

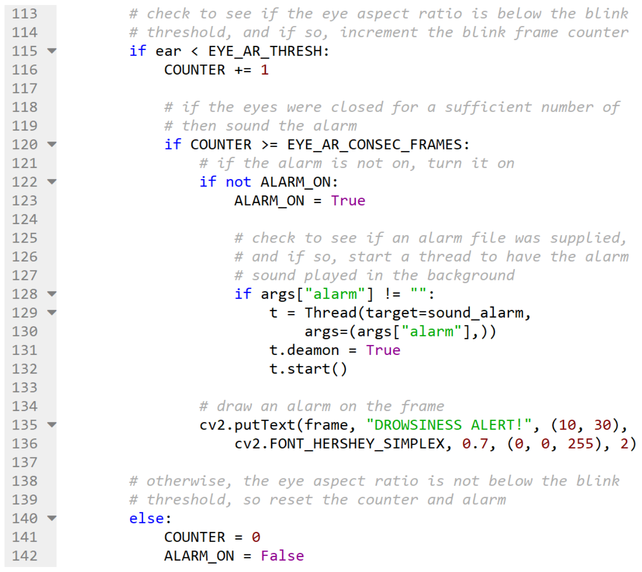
The loop which beings in line 89 in Figure 15 applys dlib's facial landmark detector (line 93) for each of the identified faces. It also converts the result to a NumPy array (line 94) and extracts the (x, y)-coordinates of the left and right eye (line 98 and 99). Hereupon their eye aspect ratios will be calculated (line 100 and 101). Then the average of the EAR of both of the eyes will be computed (line 104) [17].
The function *cv2.drawContours* (line 110 and 111) in Figure 15 mark the eyes with frames [17].



```python
88          # loop over the face detections
89          for rect in rects:
90              # determine the facial landmarks for the face region, then
91              # convert the facial landmark (x, y)-coordinates to a NumPy
92              # array
93              shape = predictor(gray, rect)
94              shape = face_utils.shape_to_np(shape)
95
96              # extract the left and right eye coordinates, then use the
97              # coordinates to compute the eye aspect ratio for both eyes
98              leftEye = shape[lStart:lEnd]
99              rightEye = shape[rStart:rEnd]
100             leftEAR = eye_aspect_ratio(leftEye)
101             rightEAR = eye_aspect_ratio(rightEye)
102
103             # average the eye aspect ratio together for both eyes
104             ear = (leftEAR + rightEAR) / 2.0
105
106             # compute the convex hull for the left and right eye, then
107             # visualize each of the eyes
108             leftEyeHull = cv2.convexHull(leftEye)
109             rightEyeHull = cv2.convexHull(rightEye)
110             cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
111             cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)
```

Figure 15: Localizing the important regions of the face and drawing frames around the eyes [17].

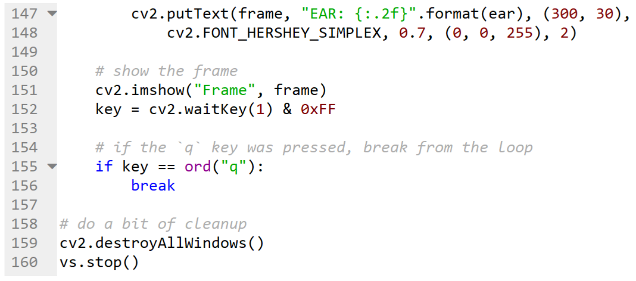
Checking the EAR if its below the threshold (Figure 16, line 115). The system start counting the number of frames if the EAR falls below this threshold from 0.3. If the car driver has closed their eyes for over 48 frames, which means the car driver not only blinks, he or she falls asleep (*EYE_AR_CONSEC_FRAMES*, line 120), the alarm sound will play. The *COUNTER* is defined in line 57 and count the number of consecutive frames where the eyes are closed. If the *COUNTER* pass the *EYE_AR_CONSEC_FRAMES* and the alarm sound is not activated (line 122), the alarm sound will play (line 123). The alarm sound is played by starting a thread (line 128 to 132). On the frame will be notified *"DROWSINESS ALERT"* (line 135 and 136). If the eyes are open, the counter will reset and the alarm sound will be off (lines 140 to 142) [17].



```
113         # check to see if the eye aspect ratio is below the blink
114         # threshold, and if so, increment the blink frame counter
115         if ear < EYE_AR_THRESH:
116             COUNTER += 1
117 
118             # if the eyes were closed for a sufficient number of
119             # then sound the alarm
120             if COUNTER >= EYE_AR_CONSEC_FRAMES:
121                 # if the alarm is not on, turn it on
122                 if not ALARM_ON:
123                     ALARM_ON = True
124 
125                     # check to see if an alarm file was supplied,
126                     # and if so, start a thread to have the alarm
127                     # sound played in the background
128                     if args["alarm"] != "":
129                         t = Thread(target=sound_alarm,
130                             args=(args["alarm"],))
131                         t.deamon = True
132                         t.start()
133 
134                 # draw an alarm on the frame
135                 cv2.putText(frame, "DROWSINESS ALERT!", (10, 30),
136                     cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
137 
138         # otherwise, the eye aspect ratio is not below the blink
139         # threshold, so reset the counter and alarm
140         else:
141             COUNTER = 0
142             ALARM_ON = False
```

Figure 16: Checking if the person in the video stream is starting to show symptoms of drowsiness [17].

The last code block (Figure 17) describes the user interface, the output frame to the screen. On the frame will be notified the current EAR value (line 147 to 148). From line 151 to 152 the frame will be build. If the car driver would like to stop this whole program, he/she will have to press q (line 155).



```
147         cv2.putText(frame, "EAR: {:.2f}".format(ear), (300, 30),
148             cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
149 
150     # show the frame
151     cv2.imshow("Frame", frame)
152     key = cv2.waitKey(1) & 0xFF
153 
154     # if the `q` key was pressed, break from the loop
155     if key == ord("q"):
156         break
157 
158 # do a bit of cleanup
159 cv2.destroyAllWindows()
160 vs.stop()
```

Figure 17: Checking if the person in the video stream is starting to show symptoms of drowsiness [17].

5 Evaluation

5.1 Program Showcase

Start the program with the following command in the command prompt (for more information read section 4.5.3):

```
python detect_drowsiness.py --shape-predictor shape_predictor_68_face_landmarks.dat
--alarm alarm.wav --webcam 1
```

Figure 18 shows the user interface frame. The system has detected the drivers face and initiates the eyes. As proof, the eyes are framed clearly in green. At the top right is the EAR shown which value is up to 0.35. This means the eyes are open and the car driver is awake. Everything is alright.
But if the eyes are closed, which is shown in Figure 19, the EAR is at a value of 0.08. Referring to in section 4.4 shown Figure 8 this value is very low and is an indication for blinking. In this case, the eyes are closed for longer than 48 frames, which means the car driver is falling asleep. This can be seen from the label "DROWSINESS ALERT!" on the top left of the frame. If this label is shown, the alarm sound also will play.

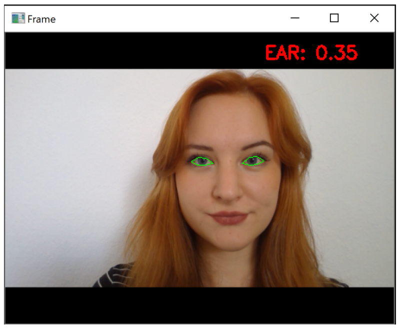


Figure 18: Output frame to the screen.

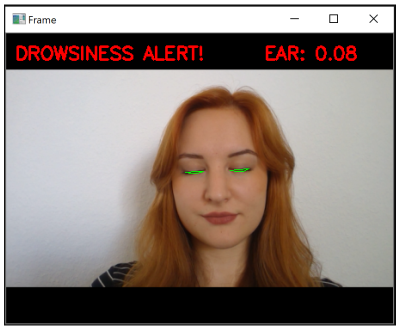


Figure 19: Output frame to the screen in a alert situation.

5.2 Implemented Requirements

Each requirement, functional and non-functional, is implemented (Figure 2). R1, R2 and R3 are performed with the help of face detection (section 4.1), facial landmark detection (section 4.2), specific facial structures detection (section 4.3) and real-time eye blink detection (section 4.4) which leads to the drowsiness detection system (section 4.5). R4 is fulfilled by section 4.5.6. If the eyes are closed over a 48 frames, the system plays the alarm sound. All of the non-functional requirements (R5, R6, R7, R8) are conform. The section 4.5 demonstrates this.

6 Summary

Towards the end, this valuation of the project is clearly useful. Detecting the drowsiness of the car driver by detecting specific facial landmarks especially the eyes and using the eye aspect ratio to analyse tells if the car driver is sleeping or not.
In the introduction (section 1) is described that it is necessary to analyse car drivers drowsiness detection and warn them because the number of accidents with personal injury due to driver fatigue in the federal territory includes many casualties and injuries.
The following section (2) displays that there are some related works with the same destination but a different approach. Only the approach from VW is similar. But the only negative side of detecting the drivers drowsiness by real-time eye blink detection is, in my opinion, that the system has to be adjusted individually. The EAR values are different from driver to driver, but the stance on car seat, too.
Next section, the concept, defined the goal of the project, the requirements and the use cases.
The implementation section (4) describes in detail what kind of detection tools the tool uses and how these are implemented in the program. For using the real-time eye blink detection (section 4.4) the program employs HOG face detection, facial landmark detection, the iBUG 300-W data set for specific facial structure detection and also the eye aspect ratio values to analyze, if the eyes are open or closed.
The evaluation (section 5) demonstrate, that the program works and meets all demands. The showcase displays the user interface frame.
On the whole the program works and serve the purpose. It would be operative to implement this into real cars or into other vehicles like buses, motor lorries or trains, to prevent accidents. Human life will be saves if this program is implement into vehicles. If you have this system, which detects your drowsiness and warns you if you are sleepy or falls asleep, in your vehicle, you will safe the other road users and you will safe yourself in case of driving while being drowsy.

References

[1] Škoda Auto a.s. *Childishly Simple: Škoda Assistance Systems*. 2020. URL: https://www.skoda-storyboard.com/en/innovation/childishly-simple-skoda-assistance-systems/. (accessed: 11.06.2020).

[2] Mercedes-Benz - A Daimler Brand. *ATTENTION ASSIST: Drowsiness-detection system warns drivers to prevent them falling asleep momentarily*. 2020. URL: https://media.daimler.com/marsMediaSite/en/instance/ko.xhtml?oid=9361586&relId=1001&resultInfoTypeId=172#toRelation. (accessed: 11.06.2020).

[3] Statistisches Bundesamt. "Verkehr: Verkehrsunfälle". In: *Fachserie / 8 / 7 / Monatlich* 2019, 2 - 2020, 2 (2019 - 2020), p. 41.

[4] L. Chen, M. Ali Babar, and B. Nuseibeh. "Characterizing Architecturally Significant Requirements". In: *IEEE Software* 30.2 (2013), pp. 38–45.

[5] N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 886–893 vol. 1.

[6] Joseph H. *The Basics of Indexing and Slicing Python Lists: A guide for beginners, by a beginner*. 2019. URL: https://towardsdatascience.com/the-basics-of-indexing-and-slicing-python-lists-2d12c90a94cf. (accessed: 21.06.2020).

[7] T. Jan et al. *Don't sleep and drive: VW's fatigue detection technology*. 2005. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.110.6980&rep=rep1&type=pdf. (accessed: 11.06.2020).

[8] V. Kazemi and J. Sullivan. "One millisecond face alignment with an ensemble of regression trees". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1867–1874.

[9] Davis King. "Correctly Mirroring Datasets". In: *Dlib C++ Library* (2018). URL: http://blog.dlib.net/2018/01/correctly-mirroring-datasets.html.

[10] Davis King. "Easily Create High Quality Object Detectors with Deep Learning". In: *Dlib C++ Library* (2016). URL: http://blog.dlib.net/2016/10/easily-create-high-quality-object.html.

[11] Davis E. King. "Max-Margin Object Detection". In: *CoRR* abs/1502.00046 (2015). arXiv: 1502.00046. URL: http://arxiv.org/abs/1502.00046.

[12] M. Köstinger et al. "Annotated Facial Landmarks in the Wild: A large-scale, real-world database for facial landmark localization". In: *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. 2011, pp. 2144–2151.

[13] Stephan Kraus. *Preventing microsleep: Bosch Driver Drowsiness Detection: Now also featured in the Volkswagen Passat Alltrack*. 2012. URL: https://www.bosch-presse.de/pressportal/de/en/bosch-driver-drowsiness-detection-41616.html. (accessed: 11.06.2020).

[14] Alfred Nischwitz and Max Fischer. *Bildverarbeitung:* 4. Auflage. Springer eBook Collection. Wiesbaden : Springer Vieweg, [2020]. URL: `http://campusapp08.hshl.de/978-3-658-28705-4.pdf`.

[15] S. Robertson and J. Robertson. *Mastering the Requirements Process: Getting Requirements Right.* Pearson Education, 2012. ISBN: 9780132942843. URL: `https://books.google.de/books?id=yE91LgrpaHsC`.

[16] Adrian Rosebrock. *Detect eyes, nose, lips, and jaw with dlib, OpenCV, and Python.* 2017. URL: `https://www.pyimagesearch.com/2017/04/10/detect-eyes-nose-lips-jaw-dlib-opencv-python/`. (accessed: 21.06.2020).

[17] Adrian Rosebrock. *Drowsiness detection with OpenCV.* 2017. URL: `https://www.pyimagesearch.com/2017/05/08/drowsiness-detection-opencv/`. (accessed: 21.06.2020).

[18] C. Sagonas et al. "300 Faces in-the-Wild Challenge: The First Facial Landmark Localization Challenge". In: *2013 IEEE International Conference on Computer Vision Workshops.* 2013, pp. 397–403.

[19] R. Sayed and A. Eskandaria. "Unobtrusive drowsiness detection by neural network learning of driver steering". In: *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering* 215.9 (2001), pp. 969–975. DOI: `10.1243/0954407011528536`. URL: `https://doi.org/10.1243/0954407011528536`.

[20] Tereza Soukupová and Jan Cech. "Real-Time Eye Blink Detection using Facial Landmarks". In: 2016.

[21] Deutscher Verkehrssicherheitsrat. *Vorsicht Sekundenschlaf!": Kampagne klärt über die Gefahren von Müdigkeit am Steuer auf.* 2020. URL: `https://www.dvr.de/programme/kampagnen/vorsicht-sekundenschlaf/`. (accessed: 10.06.2020).

[22] José Vicente et al. "Drowsiness detection using heart rate variability". In: *Medical Biological Engineering Computing* 54 (2016), pp. 927–937. DOI: `10.1007/s11517-015-1448-7`. URL: `https://doi.org/10.1007/s11517-015-1448-7`.

[23] Volkswagen. *Technology: Driver Alert System.* 2020. URL: `https://www.volkswagen-newsroom.com/en/driver-alert-system-3932`. (accessed: 11.06.2020).