In-Vehicle Drowsiness Detection

Article · January 2022

CITATIONS
0

READS
80

2 authors, including:

Giulio Federico
Università di Pisa
2 PUBLICATIONS **0** CITATIONS

SEE PROFILE

In-Vehicle Drowsiness Detection

Giulio Federico, Gianmarco Petrelli

Abstract-**When driving a vehicle, it is often challenging for someone to force his condition to keep driving even though in sleepy condition, this can likely and eventually cause traffic accident. One of the characteristics of drowsy drivers is the eyes are closed for a certain period. This work proposes an implementation of a drowsiness detection system able to dynamically detect when the state of sleepiness is increasing, thus can alert the drowsy driver. The aim of this work is to improve the effectiveness and responsiveness of drowsiness detection systems based on computer vision algorithms. A new index of drowsiness, RPERCLOS, is proposed as an evolution of percentage of eye closure (PERCLOS) on which state-of-art research in this field is currently based.**

I. Introduction

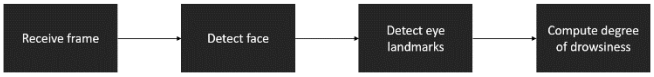


Figure 1: workflow pipeline

The pipeline of the workflow that will be executed at a high level involves the reception of a frame, to capture only the face and if this exists, calculate the landmarks of the face. All landmarks except those relating to the left and right eye will be filtered out. With these remaining landmarks, the EAR (*Eye Aspect Ratio*) will be calculated on each eye and the average value will be taken. With this value we would be able to classify if the eye is closed or open by checking if the average EAR value is below a certain threshold. Finally, the PERCLOS will be calculated in order to establish a percentage of the driver's drowsiness. This pipeline can be summarized in Figure 1, while the final result of the elaboration on a frame is visible in Figure 2.

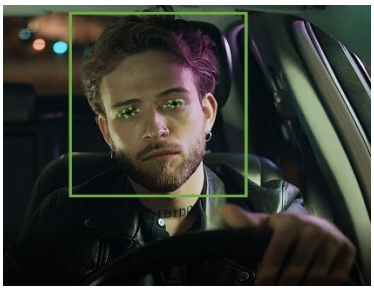


Figure 2: face detection and relative landmarks calculated on the frame

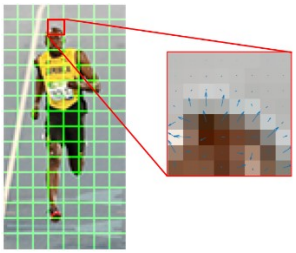
II. The Logic Behind the Algorithms

In order to better contribute to the current state of the art on drowiness detection, we decided to investigate the two algorithm approaches used in AI respectively for face and landmark detection. We think a deep understanding of what's behind it can allow us to further improve the system.

A. Face Detection

For face detection the algorithm used in our systems is the one offered by the dlib library called **dlib.get_frontal_face_detector()**. It characterizes the image (the frame) by describing the entire image through **HOG** (*Histogram of oriented gradients)*, and classifying each component found as a "face" or as "non-face" by means of an **SVM** (*Support Vector Machine)*, and then identifying the face box by appropriately combining all the classified components as "face".

HOG divides the image into blocks and at each block, for each internal pixel, it calculates the gradient as shown in figure 3.



$$\nabla F = \left[\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}\right]$$

Figure 3: gradient calculated on each pixel of a block

The value of the gradient obtained indicates the direction towards which there is a greater variation in color intensity. In figure 4 we can see three possible cases.

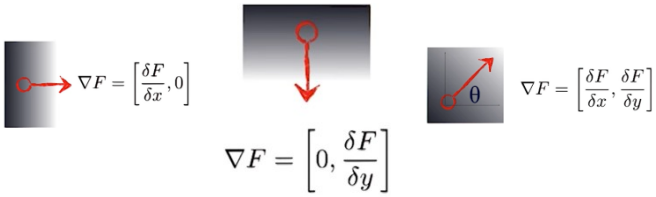


Figure 4: examples of gradients

In practice, this is done by multiplying each pixel by two *filters*, one that implements the partial derivative with respect to x and the other with respect to y.

$$H_x^D = \begin{bmatrix} -0.5 & \mathbf{0} & 0.5 \end{bmatrix} \qquad H_y^D = \begin{bmatrix} -0.5 \\ \mathbf{0} \\ 0.5 \end{bmatrix}$$

Once these gradients have been found for each pixel, each block will be characterized with a signature, that is a histogram whose bins are related to precise orientations. Depending on the orientations of its pixels each block will have a different histogram as shown in figure 5.

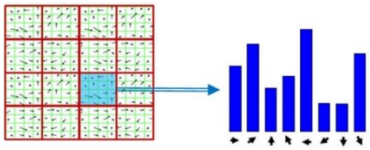


Figure 5: the histogram of a block

Once this is done for each block of the image, the concatenation of these histograms will define the final signature of the entire image as shown in figure 6:

Figure 6: the signature of the current image (frame)

Each of these blocks, together with its signature, will then be fed to an SVM who will classify the class in a binary way, i.e if this block is a "face" (or has a part of a face) or is “not a face” as shown in figure 7:

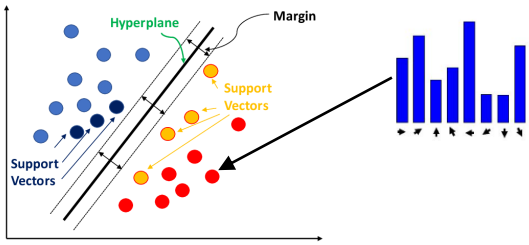


Figure 7: SVM classifier on the signatures of each block

Finally, each block classified as a "face" will be suitably assembled together with the others to define the box that identifies the face.

B. Landmarks detection

For eye landmark detection we use a method offered by dlib called **dlib.shape_predictor (“path_to_shape_predictor_68_face_landmarks.dat”)**. This method returns a predictor capable of identifying 68 landmarks on the face, i.e on the output obtained from the previous phase. The 68 landmarks are shown in figure 8:

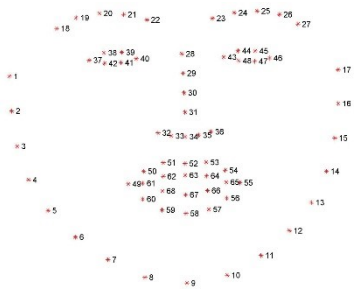


Figure 8: landmarks

The predictor used in this library is an implementation of paper [1]. It is trained using a dataset such as the **iBUG 300-W dataset** in which each image is described by an xml file inside which we find the relative coordinates for each landmark (ground truth) as shown in figure 9.

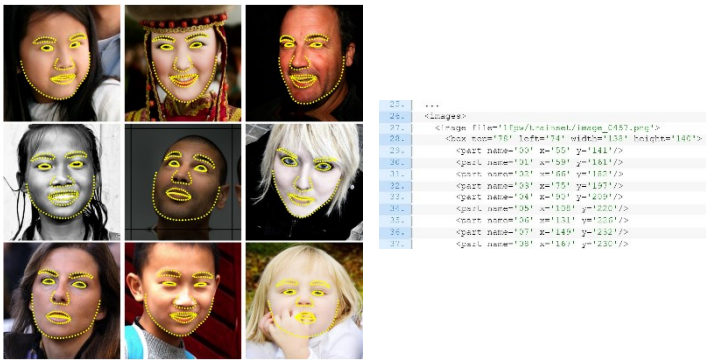


Figure 9: iBUG 300-W dataset and an example of xml file

It all begins with what was obtained in the previous phase, that is a box that outlines the region of the identified face.

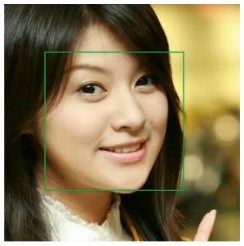


Figure 10: output result from the face detection phase

A *mean shape* is inserted in this region. A mean shape is simply a shape obtained by taking the average, for each coordinate of each landmark, of all the shapes present in the dataset. The result is a initial shape or *initial guess* as show in figure 11.

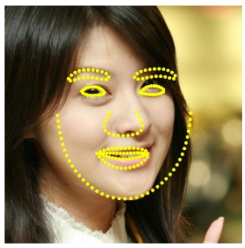


Figure 11: mean shape

At the center of the mean shape a central point is taken which will be the center of a region within which approximately 400 pixels will be taken randomly (in some versions they are taken close to the landmarks). Each of these pixels is defined by an intensity value.

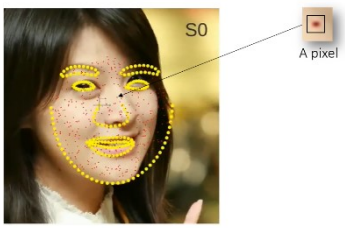


Figure 12: 400 pixels (in red) taken randomly

The mean shape was previously named "initial guess". This is the classic terminology used in boosting algorithms such as **Gradient Boost Regression Tree (GBRT)**. In fact, it is precisely this sequence of binary decision trees that will be used to transform the mean shape into a more accurate shape. In fact, each decision tree works by reducing the residuals obtained from previous trees. Each of them is usually a tree with 4-8 levels and in each level it will take as input only two pixels among the 400 taken and depending on whether their difference will be less (or equal) or greater than a certain threshold a decision will be made rather than another. One of the decision trees used in the GBRT is shown in figure 13.

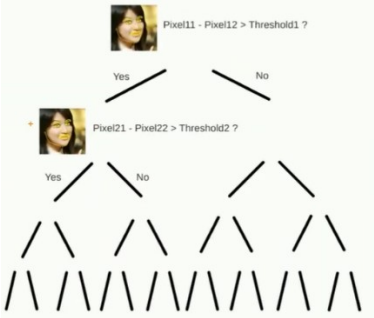


Figure 13: one of the decision trees used in the GBRT

The leaves of each tree (thus the final prediction of each of them) is the value of a vector called the *delta landmark*. This vector is a value to be added, once it's scaled by a *learning rate factor* γ, to the (x, y) coordinates of each landmark belonging to the initial guess:

$$v^1 = v^0 + \gamma \cdot \Delta_{landmark}$$

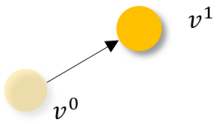


Figure 14: transformation of the position of a landmark

Since there are several (n) decision trees in sequence, each with its own delta landmark, the final coordinate of each landmark will be equal to the contribution of each of them:

$$v^1 = v^0 + \sum_{h=1}^{n} \gamma \cdot \Delta_{landmark_h}$$

Performed these transformations on each of the landmarks of the mean shape we obtain a new more accurate shape as shown in figure 15.

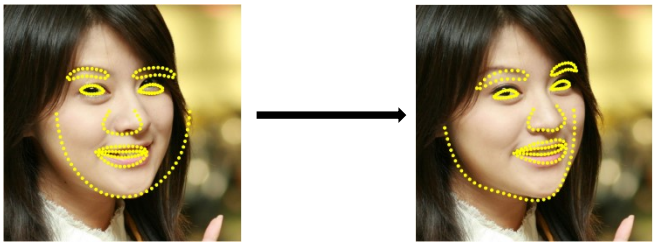


Figure 15: the new shape obtained transforming the mean shape

What has been done so far is repeated iteratively until the shape is accurately transformed. This iterative process leads to an increasingly accurate shape as shown in figure 16.

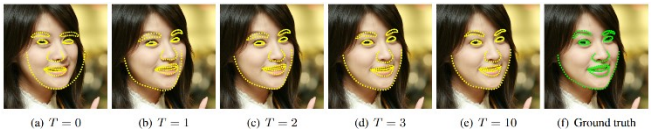


Figure 16: iterative transformation process

Of all the landmarks obtained, we will only need those relating to the left and right eye. Thanks to the **imutils** library it is easily possible to obtain the indices relating only to the landmarks of interest.

III. COMPUTE DEGREE OF DROWSINESS

The calculation of the driver's degree of drowsiness consists of two steps: calculation of the EAR *(Eye Aspect Ratio)* and PERCLOS *(Percentage Eye Closure)*.

A. EAR

EAR is a measure of how closed the eye is. Instead of classifying the eye as "open" or "closed" by simply establishing whether the eye is "totally open" or "totally closed", we calculate that measurement and decide a threshold below which we establish that the eye is "closed enough" so that some drowsiness can be detected.

We calculate the EAR on the left eye and then on the right eye. In the end we make the average. The calculation of each EAR involves taking the coordinates of the landmarks relative to that eye (Figure 17) and applying the following formula:

$$\text{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|}$$

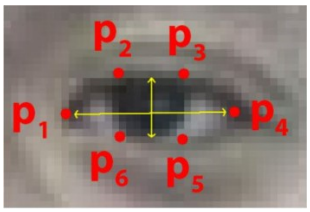


Figure 17: landmark relative to the left eye

Normally the EAR will hover in the intervals (0,0.38]. In our experiments (see later sections) the open eye EAR assumed an average of 0.34.

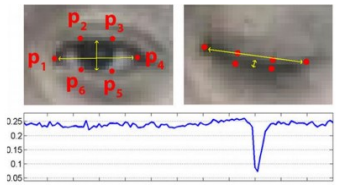


Figure 18: EAR values in the transition from "open eye" to "closed eye"

The subject of the next section will be how to calculate the EAR threshold below which eyes are considered as closed.

B. PERCLOS

The PERCLOS is a measure in percentage terms of how many times, within a time window, the EAR has gone below the allowed threshold. Its calculation is simple:

$$PERCLOS = \frac{frames\ with\ closed\ eye}{frames\ with\ closed\ eye + frames\ with\ opened\ eye} * 100$$

In our final implementation we have suggested an upgrade of this formula in order to make the system more reactive and to eliminate blinking.

In the paper [2], taking the time window of 3 minutes as the time window, the alert thresholds and their relative states are shown in figure 19:

Threshold 1	Threshold 2	Threshold 3	Threshold 4	Threshold 5
S≤3.75%	3.75%<S≤7.5%	7.5%<S≤11.25%	11.25%<S≤15%	15%<S
Low drowsiness	Low drowsiness	Moderate Drowsiness	Moderate Drowsiness	Severe Drowsiness

Figure 19: Drowsiness levels based on the perclos thresholds

IV. IMPLEMENTATION DETAILS

A. EAR fine tuning

The basic approach for defining the EAR threshold is taking a static reference value according to empirical observations. So if the (average) EAR is below this threshold then the eyes are sufficiently closed, otherwise not. However, the threshold depends a lot on the natural shape of the car driver's eye. For example, static approaches in defining this threshold would fail miserably when the driver has almond-shaped eyes as in figure 20:

Figure 20: a driver with almond-shaped eyes

Therefore our system will provide a fine tuning of 30 seconds to understand what the average EAR of the driver is. Finally, we will establish the threshold under which the EAR will be classified as "sufficiently closed eye" by taking 80% of this value. Obviously we are considering the happy case. If the driver was already tired at the start, the EAR value would be out of phase. In any case, this threshold will never be below 0.14 as calculated in [5].

B. Time Window

We need to understand the length of the time window on which the PERCLOS is calculated. In papers such as [3] this time window is 3 minutes. During our experiments we noticed that in this way the system was not very reactive. Therefore we have reduced it to 1 minute, or rather the frames that can be captured in one minute.

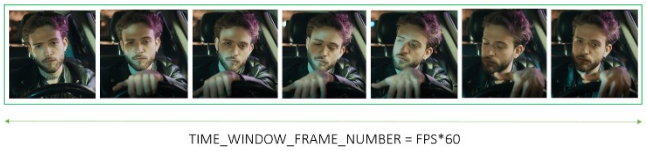


Figure 21: one-minute time window

Although the system has been tested on latest generation architectures, the main focus of this article is to analyze the performance on microprocessor systems such as Raspberry Pi 3. Therefore, every optimization in terms of memory and speed has been sought. The simple window is a *"sliding window",* i.e an array that contains only the last N frames. Each time a new frame enters the system, the one at the head of the array is deleted and the entire array is shifted to the left. Such an operation, even if using *numpy* as a (very performing) library, has an **O (n)** complexity. In general, in fact, the data of the array, with the exception of the one at the top, are copied, the new frame is inserted at the end (to the right) and a new array is then created. Instead, through the **collections** library we can implement this operation with **O (1)** complexity by moving only the pointers of the array. The *list-like container* is called **deque** and provides O (1) complexity for operations like append, pop, and shift.

There is another issue to solve. The time window has been reduced to 1 minute to make the system more responsive. While this make it really more responsive, for real time requirements this is still not enough. The reason is clearly visible by looking at figure 22.

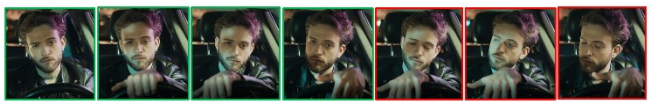


Figure 22: the last 3 frames are closed-eyes frames

Assuming a time window of only 7 frames, the PERCLOS is currently equal to $\frac{3}{3+4} \cdot 100 = 42\%$. Yet, as can be seen from figure 23, PERCLOS remains fixed at 42% even when in the best case the new frames are all open-eyed frames.

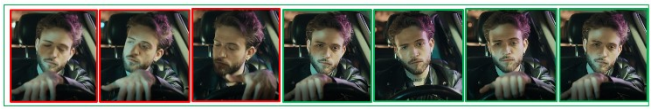


Figure 23: closed-eye frames (in red) are shifted to the head.

This means that if the time window has a period T we should wait T for the PERCLOS to return to zero. According to the paper [3] we should wait 3 minutes. In our case we just wait 1 minute. We have improved the responsiveness by considering that the "weight" that frames with eyes closed should have on the value of PERCLOS is as great as they are recent. The more time passes, the less weight these frames will have on PERCLOS. We then define a buffer of size 100 called **wake-up buffer**, handle again with deque, to record the last 100 frames with eyes closed. We record the instant in time

(through the python time library) in which these frames have historically arrived in the time window. This can be seen in figure 24.

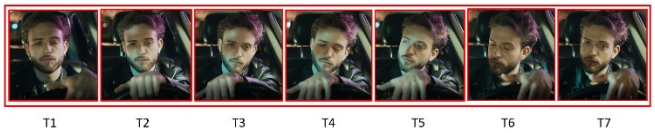


Figure 24: wake-up buffer of the last moments of time relating to frames with eyes closed

In this way, once PERCLOS has been computed, we need to multiply it with a coefficient known as the wake-up coefficient, as follows:

$$WAKE_{UP_{COEFFICIENT}} = \frac{MOVING\_AVERAGE\_WAKE\_UP\_BUFFER}{ELAPSED_{TIME}}$$

The idea behind the wake-up coefficient is to measure elapsed time with respect to the last captured frames in which eyes are closed. As a time reference, we consider the oldest timestamp present within the wake-up buffer, then we measure the simple moving average for timestamps belonging to wake-up buffer and compare it to the current timestamp. What we get is a coefficient within (0;1) able to capture most recent tiredness level of the driver.

$$\boldsymbol{R}PERCLOS = BASE_{PERCLOS} * WAKE_{UP\,COEFFICIENT}$$

C. Blinking filtering

In order to optimize the accuracy of the system in calculating the RPERCLOS index, it is necessary to evaluate, every time a new frame showing closed eyes happens, whether this is a blinking frame or not.

The idea behind the filtering task is that blinking is a very short event in time, so it is possible to recognize it and eliminate it from RPERCLOS computation.

To achieve this goal, we set a threshold, namely *total_amnt_blink_frames,* and count how many consecutive frames are classified as close; closed-eye frames are retained in moving window buffer if and only if the number of consecutive ones overcomes the threshold.

Tuning *total_amnt_blink_frames* parameter is quite challenging, to do it, we can consider that adults on average blink approximately 12 times per minute and one blink lasts about 1/3 of second.

V. SEQUENCE DIAGRAM

Below is a sequence diagram to visually clarify the steps dealt with so far.

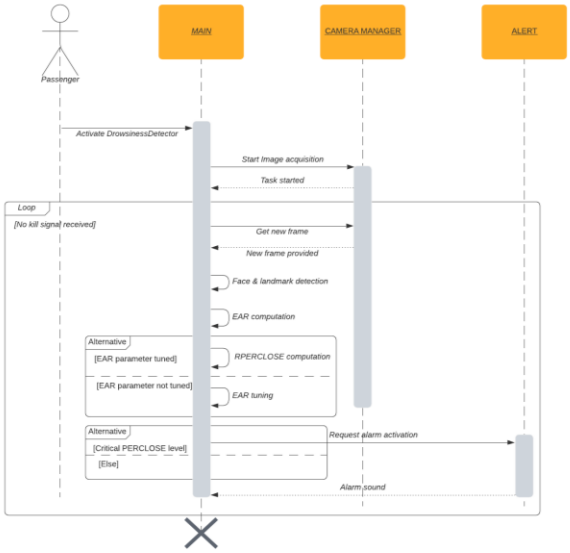


Figure 25: sequence diagram

VI. EXAMPLE OF EXECUTION

Below are three screenshots that immortalize three particular situations: one in which the user is completely awake (figure 26), one in which a certain degree of drowsiness has been detected (figure 27) and finally one in which the degree of drowsiness detected has exceeded the alarm threshold (figure 28).

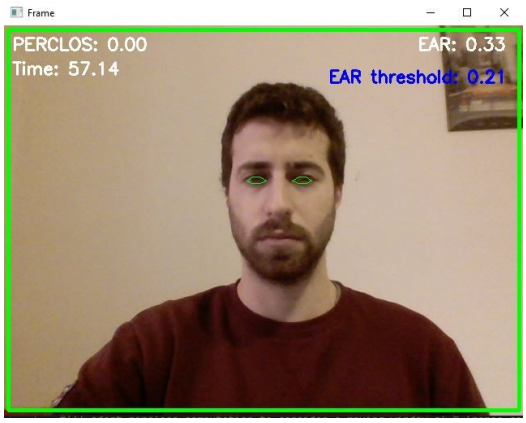


Figure 26: frame in which the state of drowsiness is null

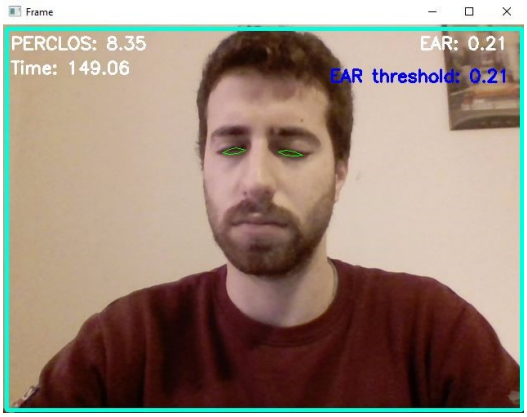


Figure 27: frame in which the state of drowsiness is growing

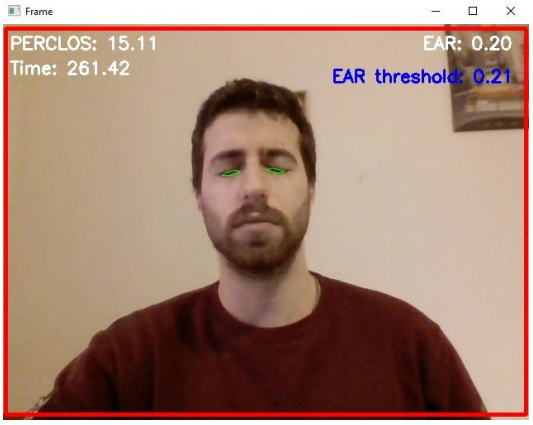


Figure 28: frame in which the state of drowsiness is beyond the threshold

VII. PERFORMANCE EVALUATION

The AI system we developed has been tested on 3 different architectures:

- Intel (R) Core (TM) i7-8750H CPU @ 2.20GHz 2.21 GHz 16.0 GB RAM (Asus ROG)
- Intel (R) Core (TM) i7-7700HQ CPU @ 2.80GHz 2.80 GHz 16.0 GB RAM (DELL XPS)
- Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz 1GB LPDDR2 SDRAM (Raspberry Pi 3 Model B+)

The main focus is to compare the first two architectures with that of the third architecture belonging to Raspberry Pi B+. The test involves the sampling of 1000 frames and the analysis of the 4 main computation times, that is:

1. face detection
2. detection of landmarks on the face
3. calculation of the EAR
4. calculation of the RPERCLOS
5. calculation of the current fps

The conducted performance analysis shows an important difference between the first two architectures and the third. The bottleneck is face detection task. The test was done with 3 possible image resolutions (1280x720, 640x480, 320x240). In both three architectures, this application can only work with the 320x240 format as the measured fps are too low on the first two resolutions. Furthermore on raspberry having only 3fps in the best case it will be very difficult to identify and eliminate the blinking. So on Raspberry this functionality has been removed.

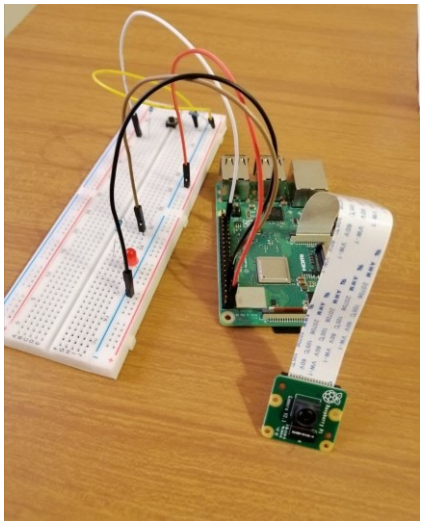
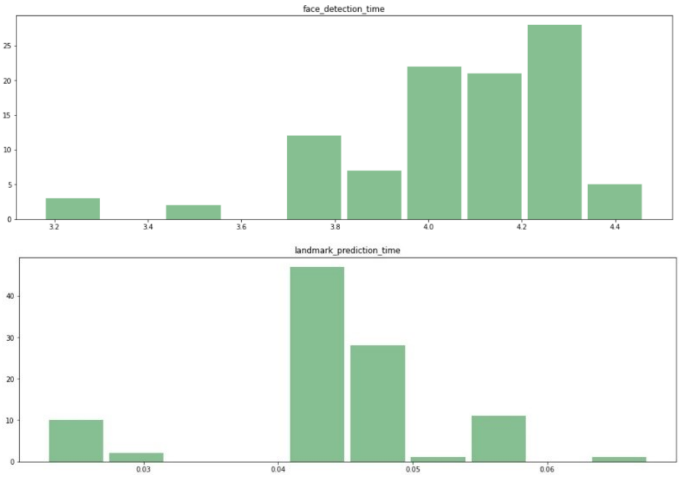


Figure 29: system prototype used to capture, process the frame (Raspberry Pi 3 Model B+) and to raise the alarm (LED red on the breadboard)

A. Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz 1GB LPDDR2 SDRAM (Raspberry Pi 3 Model B+)

	face_detection_time	landmark_prediction_time	ear_evaluation_time	percl_estimation_time	fps
mean	4.0	4.3e-02	1.6e-03	7.9e-03	0.8
mode	3.2	4.4e-02	1.4e-03	1.4e-03	1.00
median	4.1	4.4e-02	1.4e-03	6.1e-03	0.8
min	3.2	2.3e-02	6.3e-04	1.4e-03	0.51
max	4.5	6.8e-02	1.4e-02	3.1e-02	1.46



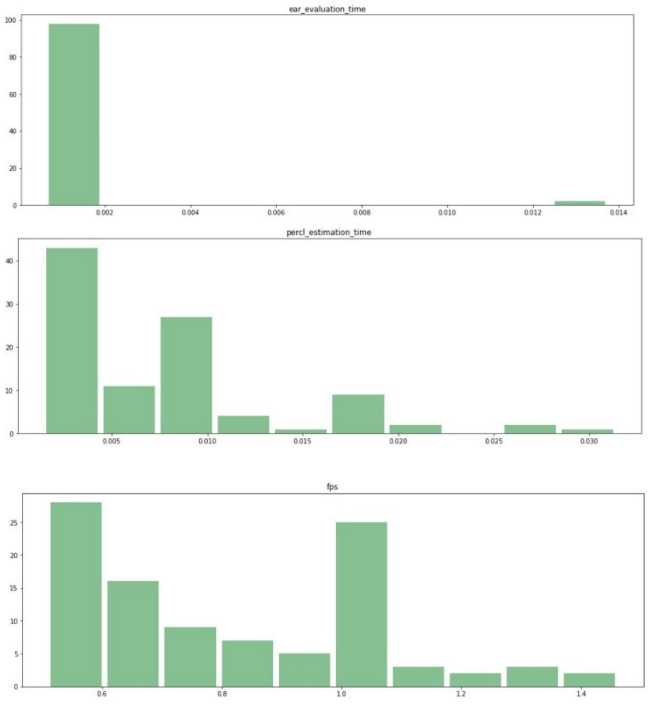


Figure 30: statistics on 100 samples with 1280x720 frames on raspberry

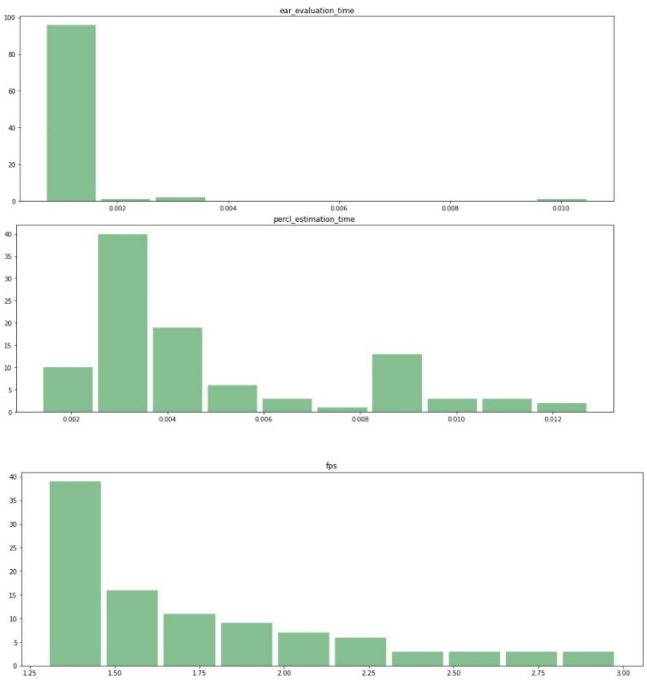
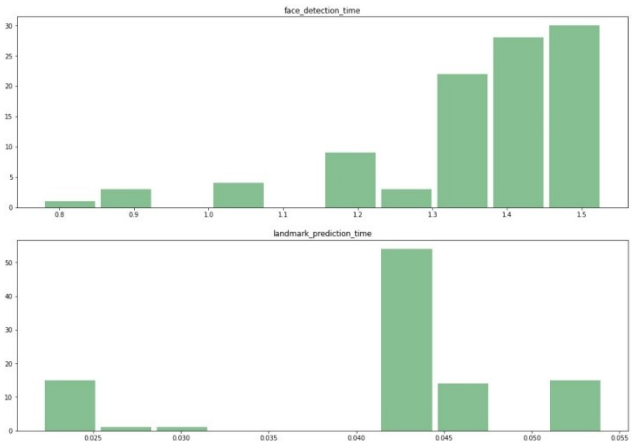
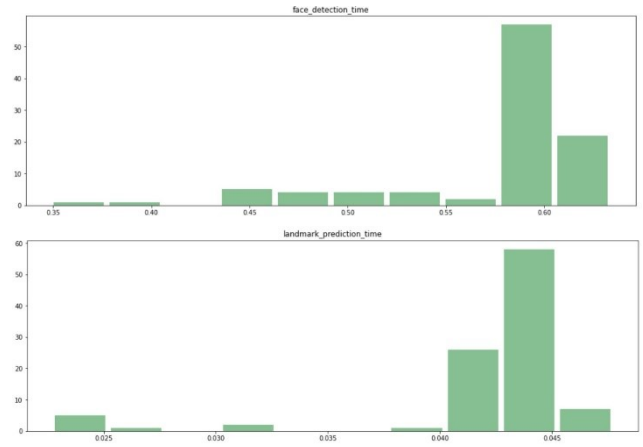


Figure 31: statistics on 100 samples with 640x480 frames on raspberry

	face_detection_time	landmark_prediction_time	ear_evaluation_time	percl_estimation_time	fps
mean	1.4	4.1e-03	1.4e-03	4.7e-03	2
mode	0.8	2.2e-02	1.4e-03	2.9e-03	1.30
median	1.4	4.3e-02	1.4e-03	3.5e-03	2
min	0.8	2.2e-02	6.8e-04	1.4e-03	1.30
max	1.5	5.4e-02	1.1e-02	1.3e-02	2.98



	face_detection_time	landmark_prediction_time	ear_evaluation_time	percl_estimation_time	fps
mean	0.6	4.2e-02	1.4e-03	8.8e-03	3
mode	0.6	4.4e-02	1.5e-03	8.2e-03	2.32
median	0.6	4.4e-02	1.4e-03	8.5e-03	2.64
min	0.4	2.3e-02	6.6e-04	3.8e-03	2.17
max	0.7	5.3e-02	4.1e-03	2.0e-02	9.95



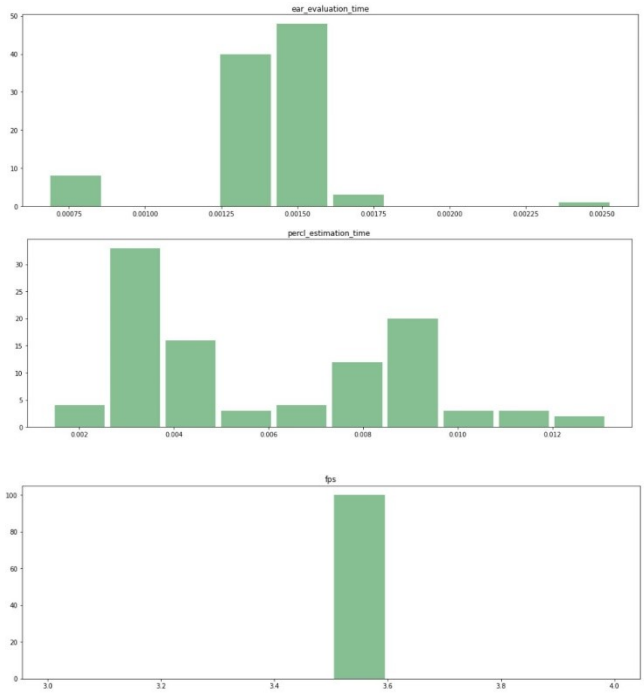


Figure 32: statistics on 100 samples with 320x240 frames on raspberry

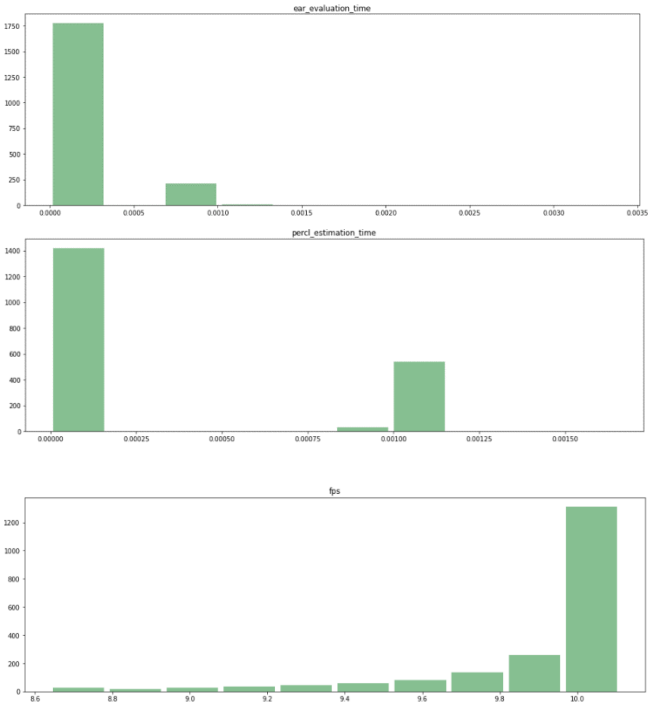
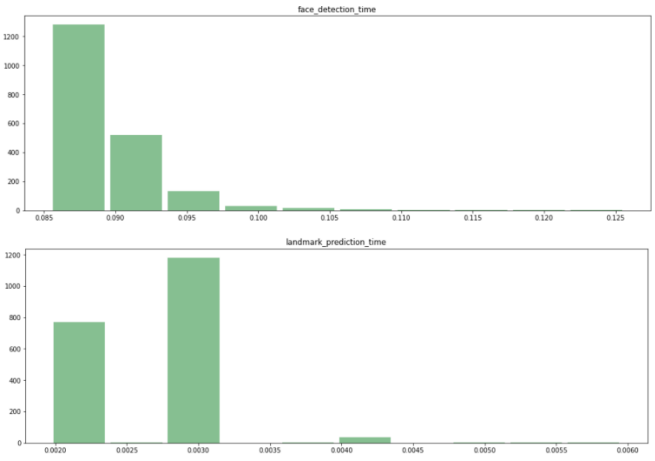


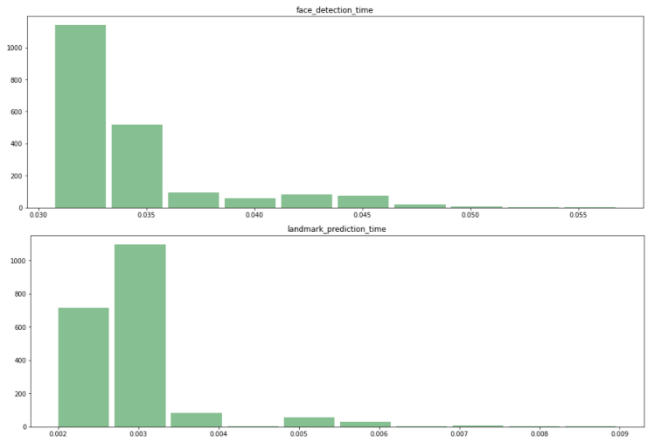
Figure 33: statistics on 1000 samples with 1280x720 frames on Asus ROG

B. Intel (R) Core (TM) i7-7700HQ CPU @ 2.80GHz 2.80 GHz 16.0 GB RAM (Asus ROG)

	face_detection_time	landmark_prediction_time	ear_evaluation_time	percl_estimation_time	fps
mean	8.9e-02	2.6e-03	1.1e-04	2.8e-04	1e+01
mode	8.7e-02	3.0e-03	0.0e+00	0.0e+00	10.10
median	8.8e-02	3.0e-03	0.0e+00	0.0e+00	10.10
min	8.5e-02	2.0e-03	0.0e+00	0.0e+00	10.00
max	1.3e-01	6.0e-03	3.4e-03	1.7e-03	9.99



	face_detection_time	landmark_prediction_time	ear_evaluation_time	percl_estimation_time	fps
mean	3.4e-02	2.8e-03	9.7e-05	2.9e-04	2e+01
mode	3.3e-02	3.0e-03	0.0e+00	0.0e+00	22.99
median	3.3e-02	3.0e-03	0.0e+00	0.0e+00	21.87
min	3.1e-02	2.0e-03	0.0e+00	0.0e+00	12.57
max	5.7e-02	9.0e-03	1.1e-03	3.4e-03	23.11



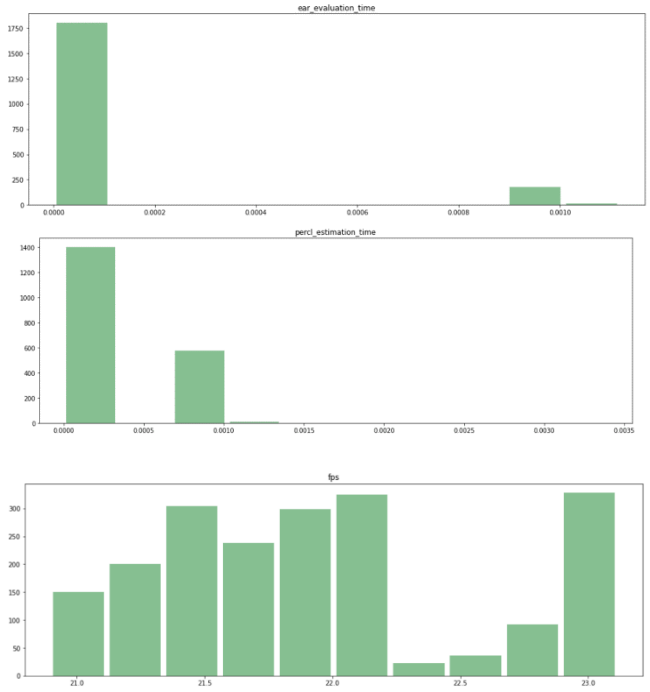
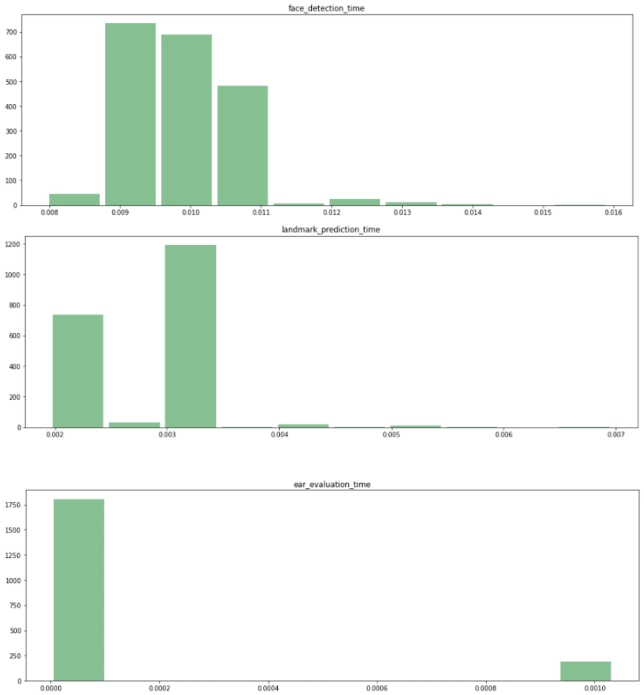


Figure 34: statistics on 1000 samples with 640x480 frames on Asus ROG

	face_detection_time	landmark_prediction_time	ear_evaluation_time	percl_estimation_time	fps
mean	9.9e-03	2.7e-03	9.4e-05	2.2e-04	5e+01
mode	9.0e-03	3.0e-03	0.0e+00	0.0e+00	52.65
median	1.0e-02	3.0e-03	0.0e+00	0.0e+00	52.60
min	8.0e-03	2.0e-03	0.0e+00	0.0e+00	18.74
max	1.6e-02	7.0e-03	1.0e-03	2.3e-03	52.71



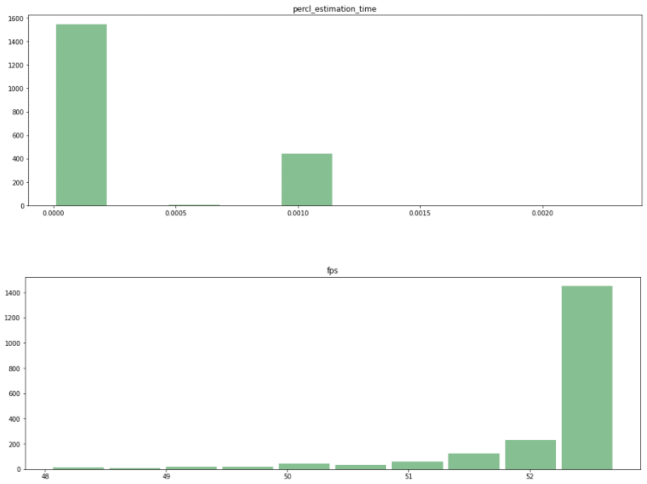
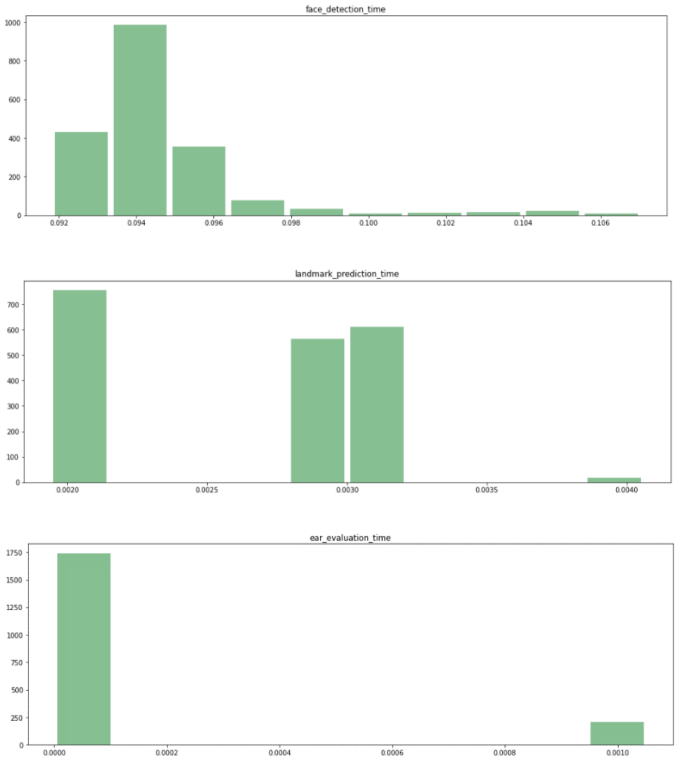


Figure 35: statistics on 1000 samples with 320x240 frames on Asus ROG

C. Intel (R) Core (TM) i7-7700HQ CPU @ 2.80GHz 2.80 GHz 16.0 GB RAM (DELL XPS)

	face_detection_time	landmark_prediction_time	ear_evaluation_time	percl_estimation_time	fps
mean	9.5e-02	2.6e-03	1.0e-04	2.5e-04	1e+01
mode	9.4e-02	3.0e-03	0.0e+00	0.0e+00	9.55
median	9.4e-02	3.0e-03	0.0e+00	0.0e+00	1e+01
min	9.2e-02	1.9e-03	0.0e+00	0.0e+00	9.22
max	1.1e-01	4.1e-03	1.1e-03	1.1e-03	9.56



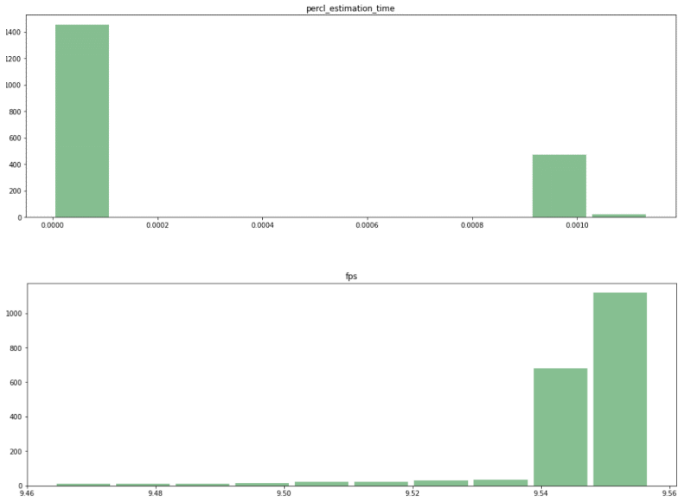
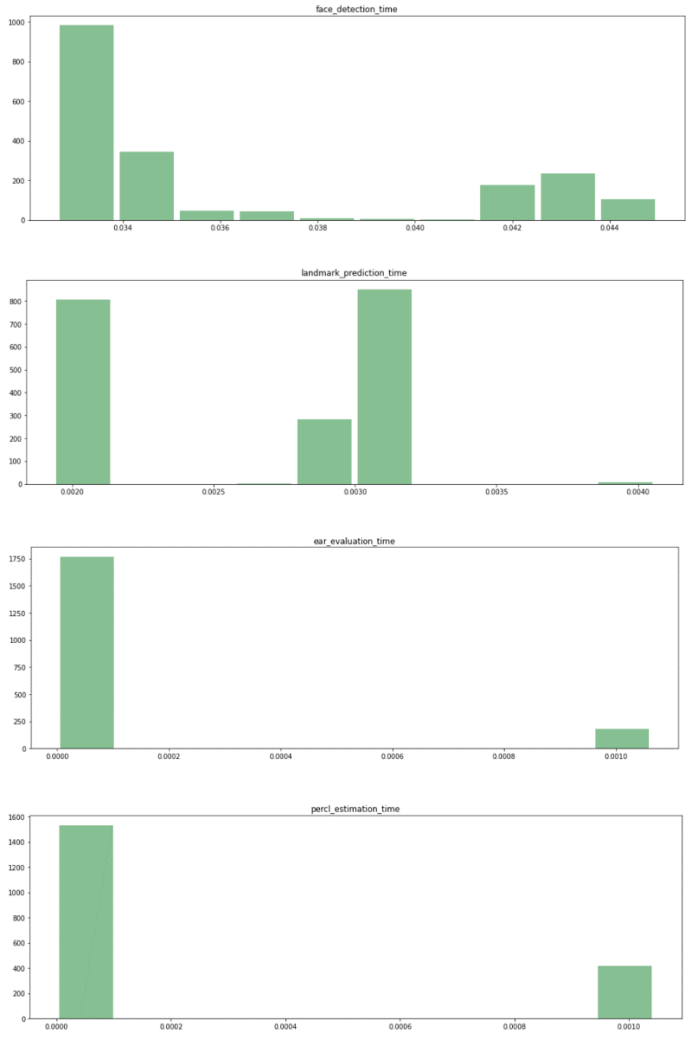


Figure 36: statistics on 1000 samples with 1280x720 frames on DELL XPS

	face_detection_time	landmark_prediction_time	ear_evaluation_time	percl_estimation_time	fps
mean	3.6e-02	2.6e-03	9.3e-05	2.1e-04	2e+01
mode	3.3e-02	3.0e-03	0.0e+00	0.0e+00	22.87
median	3.4e-02	3.0e-03	0.0e+00	0.0e+00	2e+01
min	3.3e-02	1.9e-03	0.0e+00	0.0e+00	20.33
max	4.5e-02	4.1e-03	1.1e-03	1.0e-03	22.89



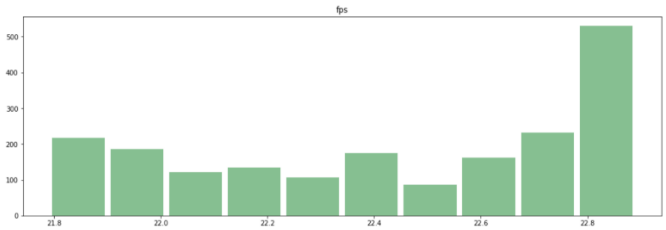


Figure 37: statistics on 1000 samples with 640x480 frames on Asus DELL XPS

	face_detection_time	landmark_prediction_time	ear_evaluation_time	percl_estimation_time	fps
mean	1.2e-02	2.7e-03	1.4e-04	2.0e-04	4e+01
mode	9.0e-03	3.0e-03	0.0e+00	0.0e+00	45.04
median	9.0e-03	3.0e-03	0.0e+00	0.0e+00	5e+01
min	8.6e-03	1.9e-03	0.0e+00	0.0e+00	40.67
max	2.2e-02	4.0e-03	1.1e-03	1.1e-03	46.04

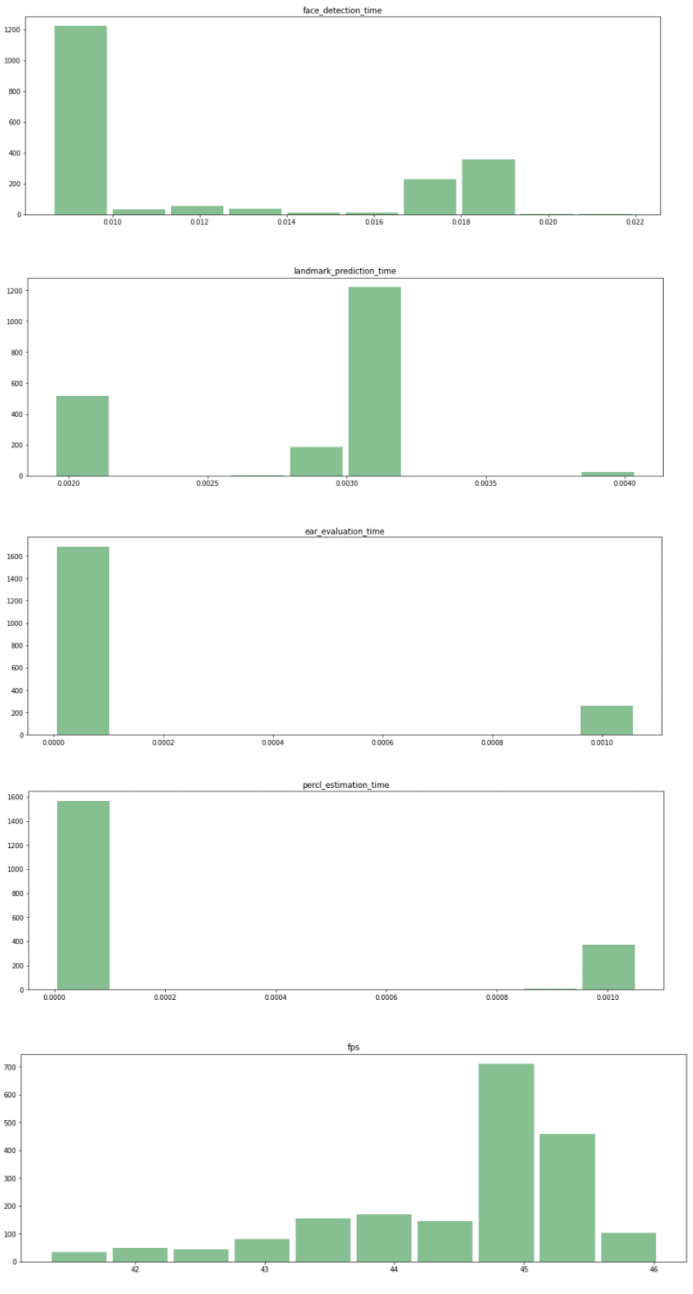


Figure 38: statistics on 1000 samples with 320x240 frames on Asus ROG XPS

REFERENCES

[1] https://www.researchgate.net/publication/264419855_One_Millisecond_Face_Alignment_with_an_Ensemble_of_Regression_Trees

[2] https://www.researchgate.net/publication/342270386_Drowsiness_Detection_Based_on_Facial_Landmark_and_Uniform_Local_Binary_Pattern

[3] https://www.researchgate.net/publication/283018835_Eye_tracking_system_to_detect_driver_drowsiness

[4] https://www.researchgate.net/publication/322250457_Real_Time_Driver_Drowsiness_Detection_Based_on_Driver's_Face_Image_Behavior_Using_a_System_of_Human_Computer_Interaction_Implemented_in_a_Smartphone

[5] https://www.hrpub.org/download/20191230/UJEEEB9-14990984.pdf