Politecnico di Milano - Department of Electronics, Information and Bioengineering

POLITECNICO
MILANO 1863

Image Analysis and Computer Vision Project

Student Lorenzo Randazzo
ID 905565

Student Luca Bianco
ID 915240

Course Image Analysis and Computer Vision
Academic Year 2019-2020

Professor Vincenzo Caglioti

August 26, 2020

Contents

1 Introduction

From a computer scientist's perspective, human beings are machines which receive input from their sensors such as ears, eyes, and skin and which interact with the world they live in through their actuators, which are their hands, feet, and so on. Their attention can be understood by analyzing the way they direct their sensors (i.e., looking at specific locations or inspecting unknown objects by touching or smelling). Moreover, as in the case of robots, examining this attention can give us hints about their state of mind and their way of reasoning.
Among the human senses, sight has an important place in today's world where we are surrounded with digital displays be it in our mobile phones, our computers, or televisions. Instead of making passive observations of the objects around, it also gives hints about what the person actively chooses to see through eye movements. Analysis of these movements, therefore, sparked great interest in research communities.
In this project we faced the challenge to improve the accuracy and the consistency of this task that was previously faced up from other students with Matlab.
Eye tracking for market research has become increasingly important. Many leading brands are using the tool, to evaluate their products, designs, advertising or even the shopping behaviour of their customers, to optimize the overall customer experience.
The goal was to show, how gaze tracking can provide assistance in the act of detecting, where the user is looking on the screen and monitoring how much he is focused in a certain area of it. It can find application in the commercial world and advertisement, by checking, if a user is looking at a certain advertisement on a web page. To do so, it was needed to overcome some challenges, like detection of face and eye features in the videoframe, their tracking in real time and the estimation of the gaze from the features.
Eye_tracking is a Matlab program that localizes the eye center and plots the estimated gaze on the screen, using a Feature-Based approach. It is thought to work in real-time with a single low-cost camera, which you can find on PCs, tablets, and smartphones.

2 State of the art

Our analysis of the state of the art was conducted on [12] and [1].
Gaze estimation is a crucial technology in many applications such as human machine interaction and cognitive processes. For example, Gaze can be applied in therapy for children with Autism Spectrum Disorders. Gaze of people is also a key issue for humaniod robots to acquire human-like intelligence. Although accurate gaze estimation can be acquired using additional hardware such as IR sources and depth sensors, the complex setup hinders its application to daily interaction situations.
In this project, was explored the gaze estimation method under a desktop environment using a single, low cost web camera. In the last three decades, active research has been carried out for eye tracking and gaze estimation. In general, gaze estimation methods can be mainly categorized into two types: namely appearance-based methods and features-based methods.
Appearance-based gaze estimation methods take the whole eye image as an input and then construct mapping functions to gaze directions or the points of regard on a screen. The first approaches proposed an adaptive linear regression method to sparsely select the training samples. The eye images are divided into many subregions and the summary of pixels intensities of each subregion is used to generalize a feature vector for mapping to the points on screen. Other approaches were the use multimodal convolutional neural networks to estimation the gaze in the wild environment, or the use of a 3D Morphable Model to the depth data captured by a depth sensor. The latter method then cropped the eye images to frontal face for gaze estimation determining the final gaze by combining frontal gaze and the head pose.
Unlike appearance-based methods, feature based methods estimate the gaze via extracting remarkable facial points such as eye corners, eye lids, pupil centers or corneal glints. The pupil center-corneal center(PCCR) based gaze estimation methods is one of the most popular feature based methods due to its simplicity and reasonable accuracy. However, these methods require an additional IR source or strong visible light sources. The first approaches were made by replacing the corneal glints with eye corners thus the light sources can be removed. A PC-EC vector is calculated for gaze estimation where the pupil centers and eye corners are manually labelled to avoid image processing errors. The further step was to combine the PC-EC vector with estimated head pose to deal with the head movement.
Although the PC-EC vector can achieve reasonable accuracy when the head keeps still, it lacks the tolerance to deal with slightly head movement, which is an important ability when combining with estimated head pose for head pose free gaze estimation.
In figure 1 below, summary and results of all the techniques analyzed in [1] are shown. Methods are grouped into categories for easier reference. HP column shows whether the technique has head pose invariance or not. Techniques allowing small head movements are denoted by $\approx$ symbol. Output column shows what type of gaze is calculated: point of gaze ($\circ$) or line of gaze ($\measuredangle$).

	Feature	Mapping	Calibration	HP	Dataset	Output	Accuracy	References	Comments
Appearance-based	—	NN	Grid	—	—	◦	1.5–4	[16, 35, 42–44]	
	—	GP	Grid	—	[98]	◦	2	[9]	
	—	GP	Grid	≈	—	◦	n/a	[37, 53]	Rigorous calib. for HP
	—	LLI	Grid	—	—	◦	0.4	[38]	IR to locate eye
	—	LLI	Grid	—	—	◦	2.4	[45]	
	—	LLI	Grid + HP	√	—	◦	2.2–2.5	[7, 29, 48, 49]	0.85° error with fixed HP
	—	LLI	Grid	√	—	∠	4.8	[8]	
	—	LLI	—	√	—	◦	3–5	[46, 47]	Incremental calibration
	—	LLI	Grid	√	[99]	∠	4	[51]	8 cameras
	—	LLI	—	—	—	◦	3.5–4.3	[10, 50]	Saliency for calibration
Feature-based	PC-EC	GP	Grid	—	—	◦	1.6	[20, 54]	
	PC-EC	LI	Grid	—	—	∠	1.2	[22]	
	PC-EC	LI	Grid	—	—	◦	n/a	[24, 55]	
	PC-EC	PI	Grid	—	—	◦	1.2	[39]	3° without chin rest
	PC-EC	LI	Grid	√	—	◦	2–5	[56]	
	PC-EC	PI	Grid	—	—	◦	2.4	[57]	
	PC-EC	PI	Grid	√	—	◦	2.3	[18]	1.2° error with fixed HP
	Several	NN	Grid	—	—	◦	1-2	[23, 58]	
	Several	NN	Grid	√	—	◦	2–7	[21]	Few tests
	EC shift	n/a	Grid	—	—	◦	3.2	[59]	
	EC shift	LI	—	—	—	◦	3.4	[60]	Hand-coded params.
	GC-CM	LI	Grid	—	—	◦	1.5	[62]	
	Several	LI	Grid	—	—	◦	3	[17]	
	Edge energy	S^3GP	Grid	—	—	◦	0.8	[14]	
	Intensity	ALR	Grid	≈	—	◦	0.6	[28, 63]	8D or 15D feats.
	Intensity	RR	Grid	—	—	◦	1.1	[31]	120D feats.
	HOG	SVR/RVR	Grid	—	—	◦	2.2	[26]	
	Several	NN	Grid	—	—	◦	3.7	[36]	Dim. reduced to 50
	CS-LBP	S^3GP	Grid	—	—	◦	0.9	[11]	Partially labelled data
	Several	Several	Grid	—	[100]	◦	2.7	[66]	Dim. reduced to 16
	Several	Several	Grid	√	[101]	◦	3.2	[67]	
	CNN	Several	Continuous	√	[68]	∠	~6	[13]	Calib. from dataset
	Segmentation	GP	Grid	—	—	◦	2.2	[30]	

Figure 1: Different approaches presented in [1]

3 Our approach and code flowchart

In this section we will describe the code from a general perspective, with the help of a flowchart. To make the model more precise, we decided to distinguish the program into two parts. In the first part the user can repeatedly run the calibration phase and in the second part, the user will be tested with a random grid, to check the accuracy of the gaze estimation. This first part can be divided in five sections: the main loop, the eye center and nose detection, the tracking, the features collection and the calibration.

3.1 Main loop

The outer level of the program is constituted by a while loop that processes all the frames arriving from the webcam. The while loop is interrupted, when the user closes the calibration or the test windows, or when the number of frames reach the established max number. For each frame, depending on the states of the program, detection or tracking is applied; after this phase, the features are collected and calibration can be performed.

3.2 Eye center and nose detection

If the number of points tracked for eyes or nose is below a certain threshold, the detection phase is started. First, the Viola-Jones [2] objects detection is used to detect the face, eye, and nose. After that, eyes center, eyelids, eyes' corners and external part of the eyes are calculated, and the tracking is initialised.

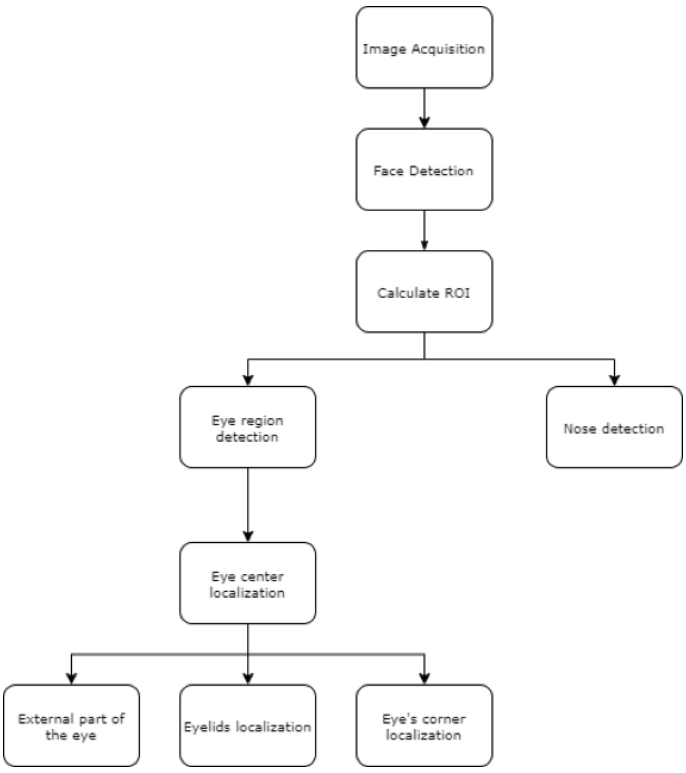

Figure 2: Detection code flow

3.3 Tracking

Once the desired points are detected, the two regions of interest for each eye plus the one for the nose are used to initialize the Kanade-Lucas-Tomasi (KLT) tracking algorithm [3], using the detection features by Shi-Tomasi, as they are implemented in Matlab; thus five trackers are used.

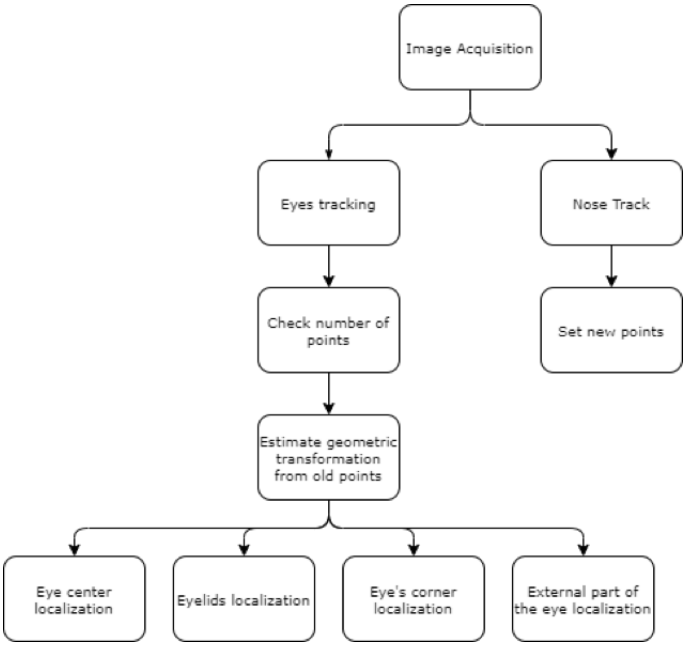

Figure 3: Tracking code flow

3.4 Features collection and prediction

Once the desired points are tracked and its real time positions are obtained, the features of interest are collected and, depending of the script version, there's the prediction (in test version) of the gaze point or the packaging of the features array (in calibration version).

3.5 Calibration

The calibration procedure is done only in the calibration version. This is performed by looking at known points on the screen and use the gathered data to calculate the coefficients of the linear regression models.

4 Detection phase

The detection is divided in different phases as shown in the flowchart: first the face is detected by means of the Viola-Jones algorithm [2]. The algorithm is available in Matlab as part of the Computer Vision Toolbox. The algorithm takes as input the gray-scale image, find the face and returns a matrix of bboxes containing all the faces found. The program selects the first one and it is not expected to handle multiple faces in the same frame. If no face is found the program just drops the frame and continue to the next iteration.
Once the face box is found, the localization of the eyes is performed in a limited Region Of Interest (ROI) defined with respect to the face box found in the previous step. For both cropped ROI images we perform the following steps. In figure 4 you can see the eye ROIs in yellow boxes and the face ROI in the red one.

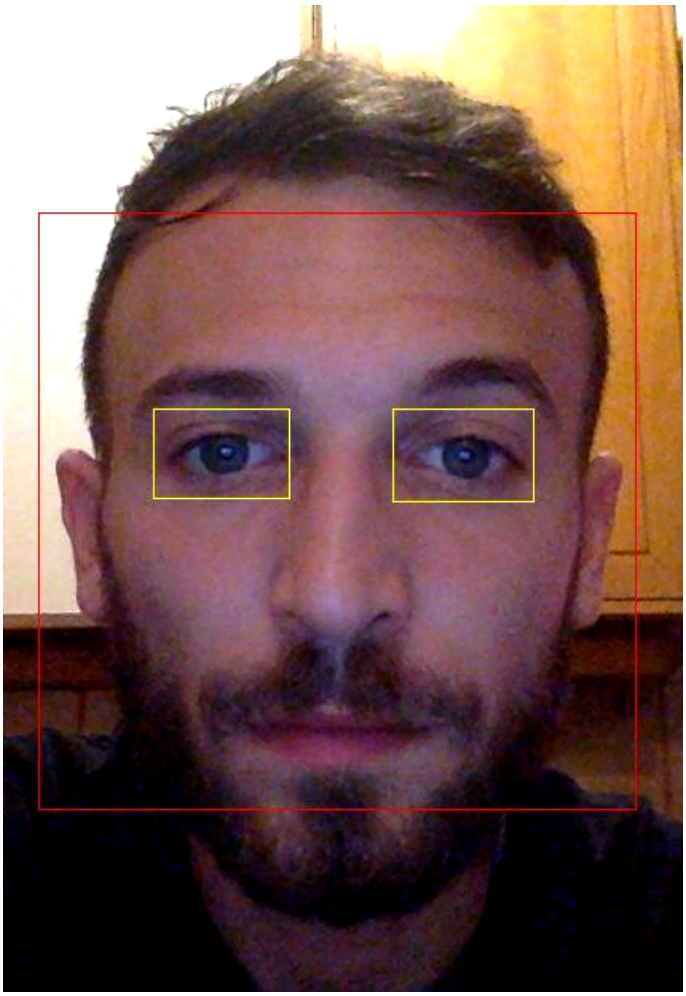

Figure 4: Head ROI used

Given the region of interest from the Viola-Jones detection of the eyes, the localization of the eye centers is performed in a few steps. The following algorithm has been taken from [4].

4.1 YCbCr color space conversion

The cropped images are first converted from RGB to YCbCr colour space. This latter colour space is ideal to distinguish the skin area from the eye area. The eye area has a blue dominant resulting in higher values in the Cb channel. The skin area has a red dominant resulting in higher values in the Cr channel. Also, the separation of the luminance information makes the skin modelling more resilient to different lighting conditions and uneven illumination. Figure 5 shows a converted image of the left eye ROI.

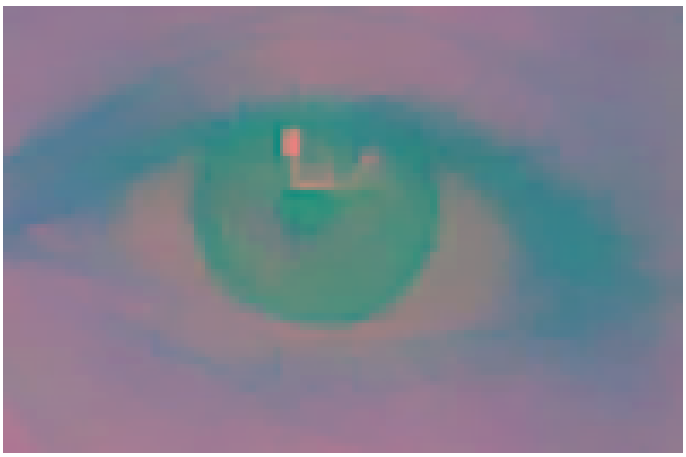

Figure 5: Left eye converted to YCbCr

4.2 EyeMapC calculation

The eyeMapC is the result of the combination of Cb and Cr channels to highlight the eye area. The eyeMapC is defined as in equation 1 where Cb, Cr are normalized to the interval $[0,1]$ and $\bar{Cr}$ means (1 - Cr). The Cb, Cr division could produce inf or NaN pixel values. inf is saturated to the max numerical value and NaN is replaced by 0. Large values on the eye map are observed at the positions of the eye regions and eyebrows where the colour difference from the skin pixels is maximized.

$$EyeMapC = \frac{1}{3}[Cb^2 + \bar{Cr}^2 + \frac{Cb}{Cr}] \tag{1}$$

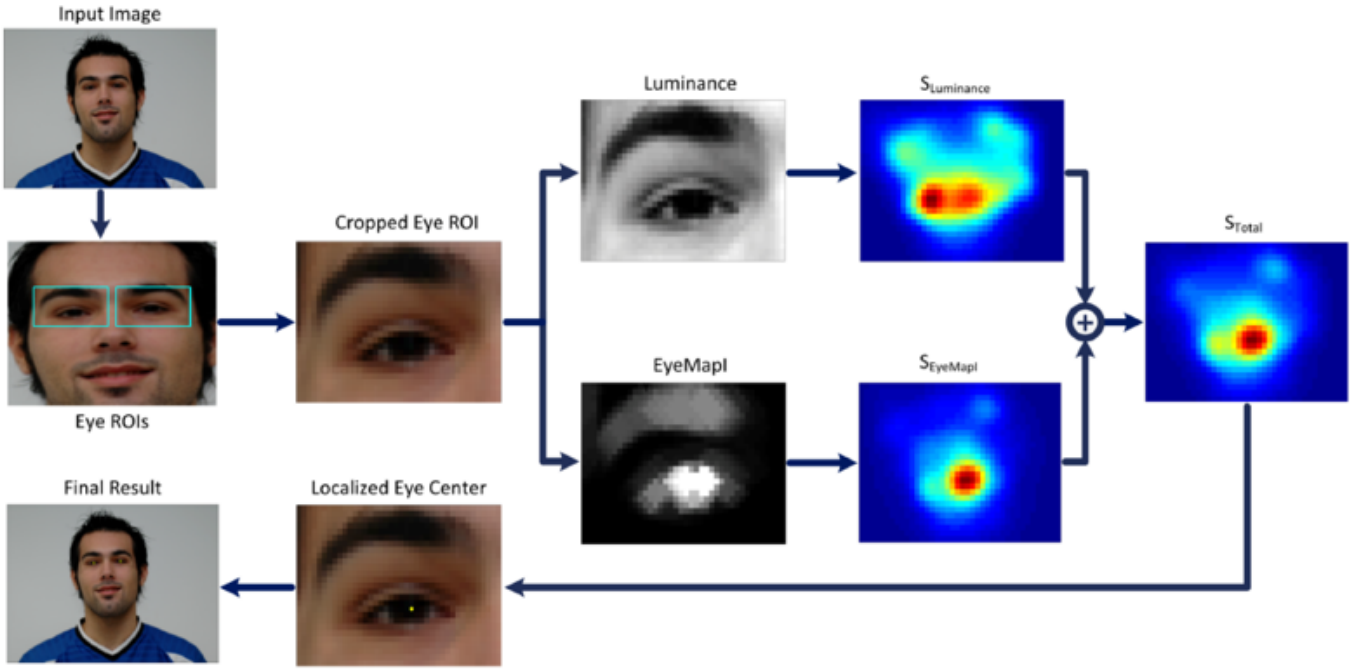

Figure 6: Overview of the proposed system. For clarity purposes, the images showing radial symmetry transform results are pseudo colored. Image taken from [11]

4.3 EyeMapI calculation

The irises present significantly lower brightness values than the sclera and the skin areas. So, we want to fuse the information of the eyeMapC with the luminance channel. Therefore, we calculate a new eye map eyeMapI, that is, the division of eyeMapC by Y, the luminance channel. The process is graphically represented in figure 7.
The eyeMapC has high values in the iris area while Y has low values in the same area, resulting in a new eye map that further enhances the iris area. Moreover, the use of morphological operations such as dilation and erosion further accentuate the irises darker appearance in the Y component and the brighter appearance in the eyeMapC component. We perform these operations with two at circular structuring elements called B1 and B2, for eyeMapC and Y respectively. The radius of these circular elements is defined with respect to the iris radius.

$$EyeMapI = \frac{EyeMapC \oplus B1}{(Y \ominus) + \delta} \tag{2}$$

where $\oplus$ and $\ominus$ denote gray-scale dilation and erosion, respectively. δ is used to solve numerical problems deriving from a zero division.

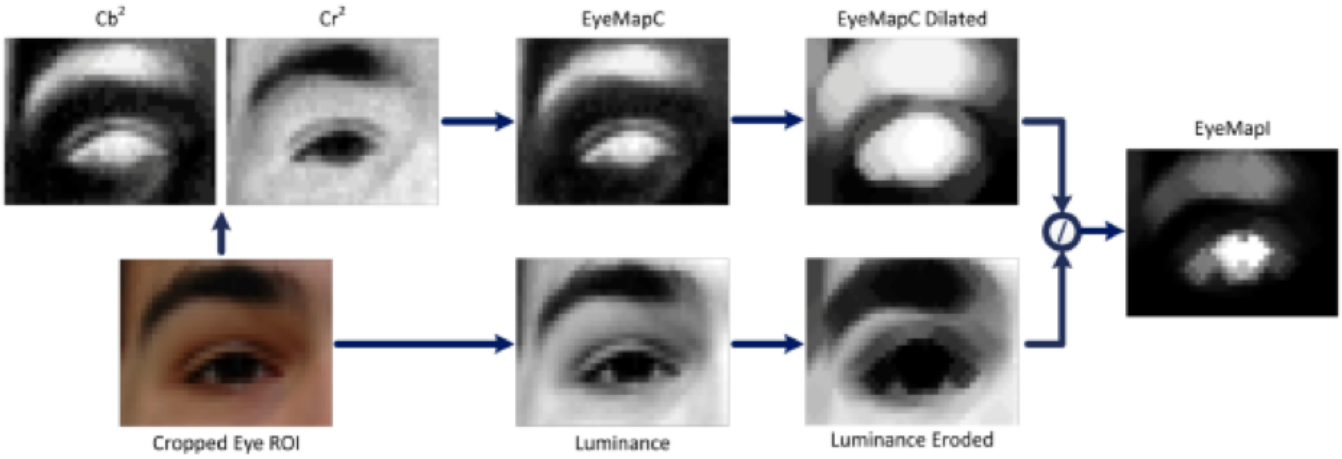

Figure 7: EyeMapI construction. Image taken from [11]

A small static number would suffice, yet, experimental tests exhibited improved results when a dynamic, data-driven value of δ is used:

$$\delta = mean(Y \ominus B2) \tag{3}$$

The fast-radial symmetry transform (FRST) [5] is a transform that utilizes local radial symmetry to highlight points of interest within a scene. Its low computational complexity and fast runtimes makes this method well-suited for real-time vision applications. The transform relies on a gradient-based interest operator that works by considering the contribution of each pixel to the symmetry of pixels around it.
The implementation we are using is a modified version of the Loy and Zelinski's approach [5] implemented by Kovesi [6]. First, the gradient of a gray-scale image is calculated computing derivatives in x and y via Farid and Simoncelli's 5-tap derivative filters. The results are significantly more accurate than Matlab's gradient function on edges that are at angles other than vertical or horizontal. This in turn improves gradient orientation estimation enormously. Then, we define a set of discrete radii N.
The fast-radial symmetry transform algorithm is applied to the eroded luminance image $YEroded$ and to the $EyeMapI$. Then, the two transforms $S_{YEroded}$,$S_{eyeMapI}$ are summed together and the position of the maximum value pixel is the position of the eye center.

$$(x_c, y_c) = argmax(S_{YEroded} + S_{EyeMapI}) \tag{4}$$

4.4 Eye's corner localization

The most common approaches consider inner and outer eye points as anchor points. Instead of tracking isolated points, here is used the more efficient alternative of tracking a rectangular image patch and consider as anchor points its center coordinates.
We built a rectangular image patch for each one of them. These two patches are located near to the inner sides of the eyes, the ones between the eye's corners and the nose. The dimensions of the patches depend on the interocular distance, that is the distance between the two detected eye centers, and the dimension of the detected eye box.
The eyes' corner are then located by finding the center of this patches. This method is reliable and offer good result. In figure 9 you can see the calculated patches in red.

Figure 8: Eyelids localization

4.5 Eyelids localization

The positions of the upper and lower eyelids provide information about the degree of eye opening [7] and greatly contribute in defining the gaze along the vertical axis. The y-positions of the eyelids correspond to the horizontal boundaries between two homogeneous areas, i.e. the iris area and skin area. The x-position of the eyelids is regarded the same as those of the corresponding eye centers. Starting from the localized eye center we define a rectangular Region of Interest (ROI) in which the eyelids are searched. The distance between the eyeball centers, also known as interocular distance, is used as the reference distance.

Assuming that the iris diameter roughly corresponds to 10% of the interocular distance, the width and height of the ROI is defined as $0.1d_{ioc}$ and $0.3d_{ioc}$ correspondingly (d_{ioc} stands for the interocular distance); each vertical side is at a distance of $0.05d_{ioc}$ from the eye center so that only the iris area (not the sclera) is enclosed, thus constituting a homogeneous area, and the distance of each horizontal side is $0.15d_{ioc}$ from the eye center, so that the eyelid boundary is certainly included, regardless of the eye state.

To detect the boundary of these distinct regions, integral projection functions are used. Image projection functions have been proven to be effective methods for extracting boundaries between different areas, representing the image by 1-dimensional orthogonal projections usually along the vertical and horizontal axes [8][9].

However, in view of the specific application, head rotations may change the boundary orientation on other directions rather that the horizontal one. To this end, the integral projection function is generalized to detect projections on different angles. Suppose $I(x, y)$ is the intensity of a pixel at the location (x, y). The integral projection along a direction θ for a rectangular area is defined as

$$IP(I, \rho) = \int_{\frac{-H}{2}}^{\frac{H}{2}} I(x_0 + \rho\cos\vartheta) - h\sin\vartheta, y_0 + \rho\sin\vartheta + h\cos\vartheta)dh \tag{5}$$

where (x_0;y_0) is the rectangle center (eye center), $\rho = 0,1, \ldots ,W$, with W being the width of the rectangle, and H represents the height of the rectangle or, equivalently, the number of pixels to be integrated for each ϱ.

Given the search ROI for the eyelids, denoted here after as $I(x, y)$, we first perform an edge detection on the cropped image, through canny filter. The integral projection function of equation 5 is computed for $I(x, y)$ with θ being the inclination of the line connecting the two detected eye centers, which

represents the rotation of the head. The y-coordinate is determined finding the first value to be above a fixed threshold; the lower side position is calculated in a similar manner.

4.6 Nose detection

The detection of the nose was really straightforward since it's integrated in the Viola-Jones algorithm [2]. With this algorithm we were able to easily track the tip of the nose, even if we had to adjust the "MergeThreshold" value to be able to obtain only the true positive value of the nose tip.

5 Tracking phase

Tracking is the process of locating a moving object or multiple objects over time in a video stream. Tracking an object is not the same as object detection. Object detection is the process of locating an object of interest in a single frame. Tracking associates detections of an object across multiple frames. So, when the detection is finished, eyes and nose need to be tracked frame by frame. To do this, we used the Matlab implementation of the Kanade-Lucas-Tomasi (KLT) tracking algorithm [3], using the detection features by Shi-Tomasi [10], as they are implemented in Matlab. We used five trackers, one for each eye center, one for each eye's corner and one for the nose. Each one of the tracker concerning the eyes is initialized using the ROI the patch previously found, while for the the nose we used the tip of the nose found in the detection phase.
As the point tracker algorithm progresses over time, points can be lost due to lighting variation, out of plane rotation, or articulated motion. To solve this problem, we do again the detection when the number of visible points is less than the fixed threshold.

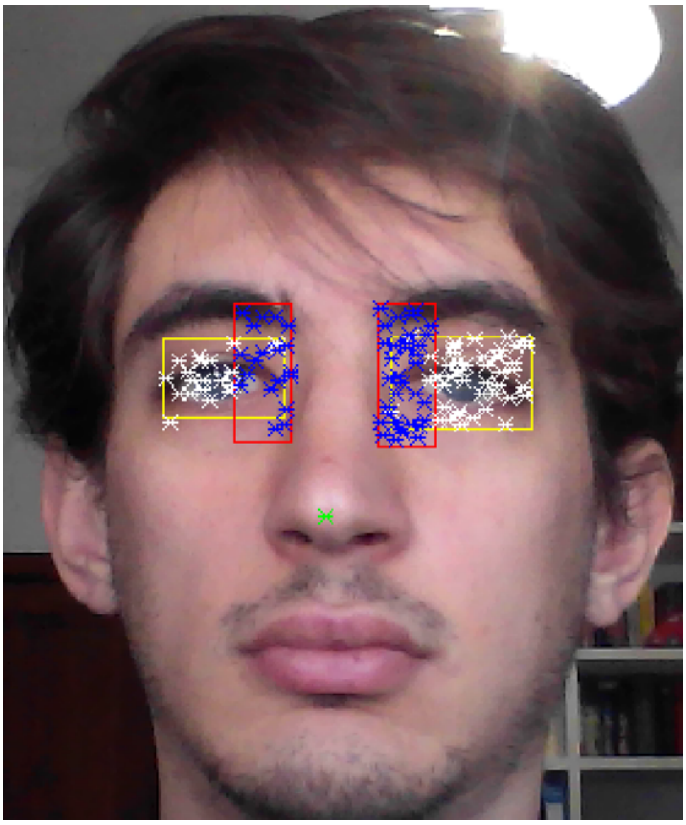

Figure 9: Points tracked in eyes region

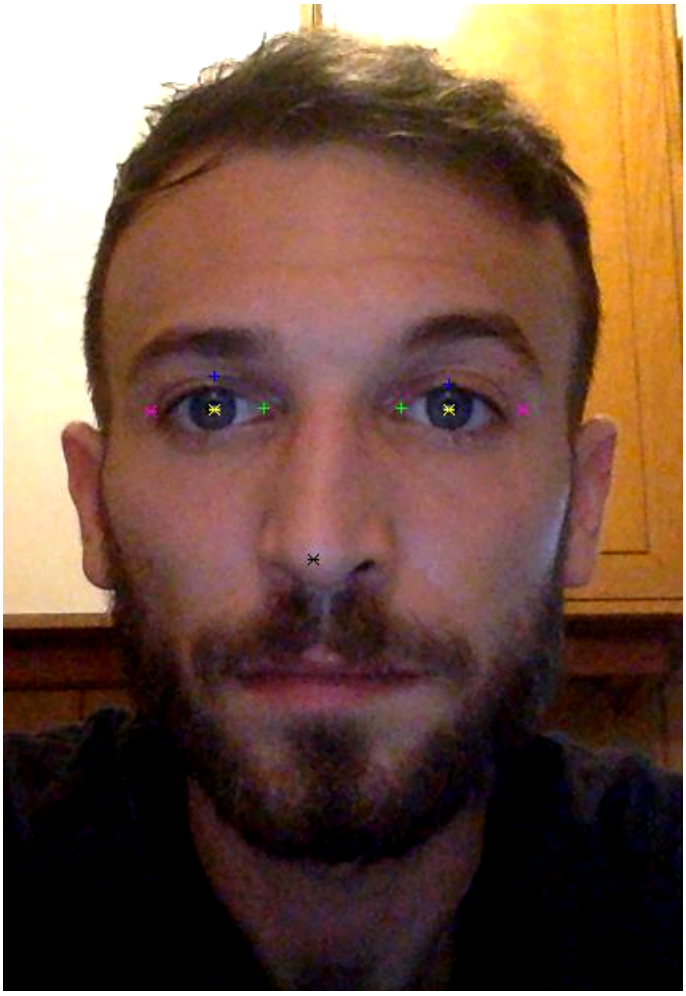

Figure 10: Points of interest in face

6 Calibration and gaze estimation

The goal of this section is to show how the prediction model is obtained, following the collection of the position of the point of interest previously described and how the calibration is performed.
The prediction model is the piece of code that map image data to screen positions, that is, the gaze point. Here a linear regression model is used. Second-order polynomial equations are commonly used for 2D mapping and are generally useful to correct curved distortions and to smooth scaling along the screen. However, nonlinear terms may introduce big errors especially when approaching to screen borders. As a result, errors during calibration can lead to much larger errors on screen coordinates estimations. Moreover, the more the coefficients to learn, the more the training examples should be obtained.
As mentioned in [12] since the features are collected with a certain distance from the screen, an error could be learned from the features when the position relative to the screen is changing through the different calibration or the test; this means that changing the position from the screen will leads to a wrong gaze estimation.
To avoid this problem, every time that a horizontal feature is collected, is then scaled by the distance between the most far points of the eyes' bbox found in detection and tracking phase, in figure 12 plotted in magenta. In the same way, the vertical features are scaled by the mean height of the two eyes' bbox, in figure 13 plotted in white. This approach leads to have better performance with different distances from the screen.
One of the features that we choose to implement is specifically used to understand whether the face is straight to the screen or if it is slightly turn on left or right. In particular, we found for each eye a parameter calculated as the difference between the most external point of the eye bbox and the nose tip. Later, the feature is calculated as the difference between this two parameter.
This is based on the assumption that, if the face is rotated to the left this value assumes a negative value, proportional to the face rotation angle. In contrast, looking to the right will result in a positive value of this feature. Moreover, when the face is frontal, we expect the feature is near zero.

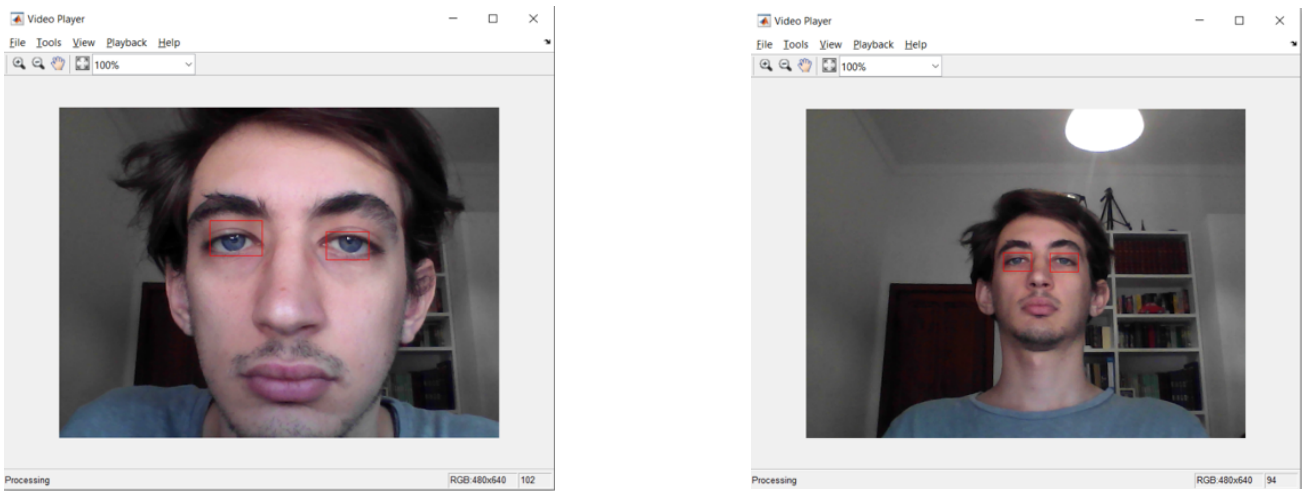

Figure 11: Eye boxes from different distances

To have a fast calibration procedure and for the reasons above explained, a linear regression approach is used.
While we were facing the problem, the common solution was to compute two mapping functions but, doing some experiment and some research we figure out that a possible approach could be to apply

the linear regression for each eye separately and then to average the two value along the x and y axes in the prediction phase. This led to a better result and more accuracy in the gaze estimation.
So, given the assumption of independence of gaze estimation in the two axes, four separate feature vectors are formed as horizontal and vertical distances between moving and anchor points, two for each eye.
Four mapping functions, one for each eye direction (along x and y axes), are learned (equations 8, 9, 10 and 11).
As mentioned before, the horizontal and vertical features are respectively scaled by the two factors h_{scale} (equation 6) and v_{scale} (equation 7).
The final points are predicted by taking the average points predicted by the regression models, as in equations 12 and 13.

$$h_{scale} = -(x_{bboxEyeLeft} - x_{bboxEyeRight} + width_{bboxEyeRight}) \tag{6}$$

$$v_{scale} = \frac{height_{bboxEyeRight} + height_{bboxEyeRight}}{2} \tag{7}$$

$$Y_{h\_left} = x_0 + \frac{h_{1left}}{h_{scale}} * x_1 + \frac{h_{2left}}{h_{scale}} * x_2 + \frac{h_{3left}}{h_{scale}} * x_3 + \frac{h_{4left}}{h_{scale}} * x_4 \tag{8}$$

$$Y_{v\_left} = x_0 + \frac{v_{1left}}{v_{scale}} * x_1 + \frac{v_{2left}}{v_{scale}} * x_2 + \frac{v_{3left}}{v_{scale}} * x_3 + \frac{v_{4left}}{v_{scale}} * x_4 \tag{9}$$

$$Y_{h\_right} = x_0 + \frac{h_{1right}}{h_{scale}} * x_1 + \frac{h_{2right}}{h_{scale}} * x_2 + \frac{h_{3right}}{h_{scale}} * x_3 + \frac{h_{4right}}{h_{scale}} * x_4 \tag{10}$$

$$Y_{v\_right} = x_0 + \frac{v_{1right}}{v_{scale}} * x_1 + \frac{v_{2right}}{v_{scale}} * x_2 + \frac{v_{3right}}{v_{scale}} * x_3 + \frac{v_{4right}}{v_{scale}} * x_4 \tag{11}$$

$$Pred_h = \frac{Y_{h\_left} + Y_{h\_right}}{2} \tag{12}$$

$$Pred_v = \frac{Y_{v\_left} + Y_{v\_right}}{2} \tag{13}$$

Features for both eyes are calculated as the following in the two axes.

Horizontal (figure 12):

- Horizontal distance between eye center and nose tip;
- Horizontal distance between eye center and eye corner;
- Horizontal distance between eye corner and nose tip;
- Difference between horizontal distances from outer eyes corner and nose tip;

Vertical (figure 13):

- Vertical distance between eye center and nose tip;

- Vertical distance between eye corner and nose tip;
- Vertical distance between upper eyelid and eye center;

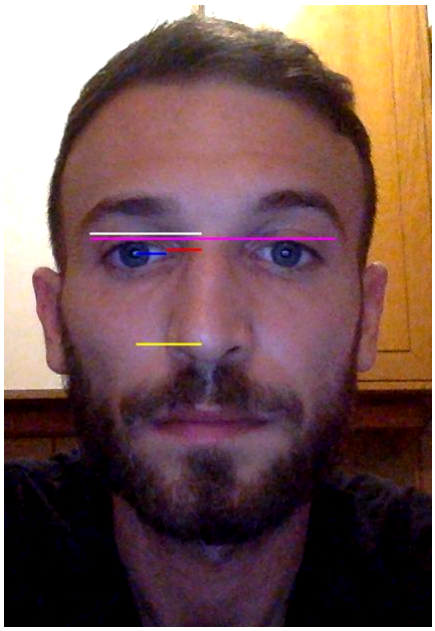

Figure 12: Horizontal features used in the estimation process

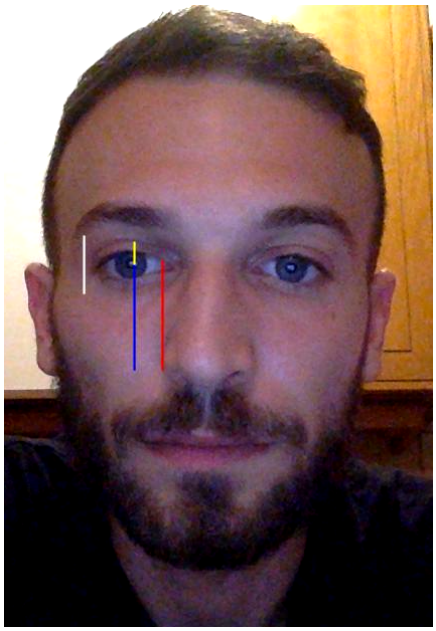

Figure 13: Vertical features used in the estimation process

After all the important points are identified in the tracking part of the program, the calibration phase is initialized. The procedure will show a window with some blue points moving on the screen.
When a point is showed the user have to look at it until the final 'bip' is heard and the point move in the next position. The calibration pattern consists in 9 points (figure 14) placed in a square arrangement. Thereafter, the correspondences between the known points on the screen and their relative features, allows to train the linear regression model.
This operation is performed by the script "gen_model.m".

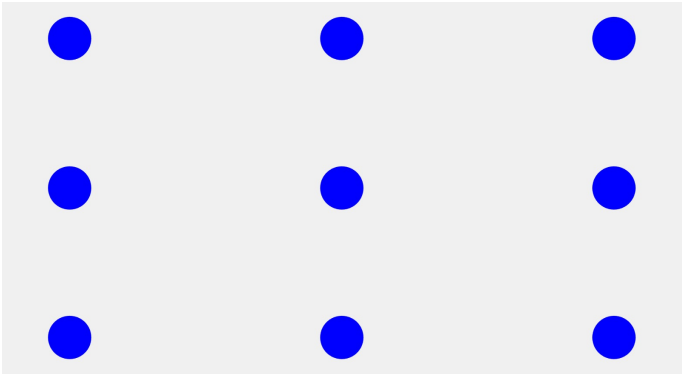

Figure 14: Calibration targets

7 Test phase

As explained before, when the calibration phase is completed, in the test phase the new input vector arrives, and it is mapped to the estimated screen point with the obtained model parameters. To approximate a real situation, in which we want to know, how precise the estimation is, we setup multiple random grids.
These grids are randomly divided in green and blue boxes, the user is asked to look at the green ones. After this phase is finished, an heatmap can be generated, showing the user, if he was able to recreate the pattern. In figures 15,16,17 and 18 are presented the results of 4 tests. On the left there is the proposed pattern, on the right there is the heatmap generated with the points looked on the screen during the test.
After this experiment, we tried to estimate the accuracy of our program, by dividing the grid in 16 different blocks and focusing on each of them for 30 seconds, one by one in different experiments. The results are listed in table 1. In each squared there is the percentage of accuracy of the corresponding square, the last column contains the accuracy of each row, the last row contains the accuracy of each column.
The mean accuracy over all the squares is 91.54%. The accuracy metric doesn't take care of the points predicted out of the screen.

92.86	98.33	95.27	82.05	92.13
92.68	95.29	84.30	89.31	90.40
97.70	92.66	89.38	95.79	93.88
94.64	92.74	75.47	96.23	89.77
94.47	94.76	86.11	90.95	91.54

Table 1: Accuracy in each square and total accuracies

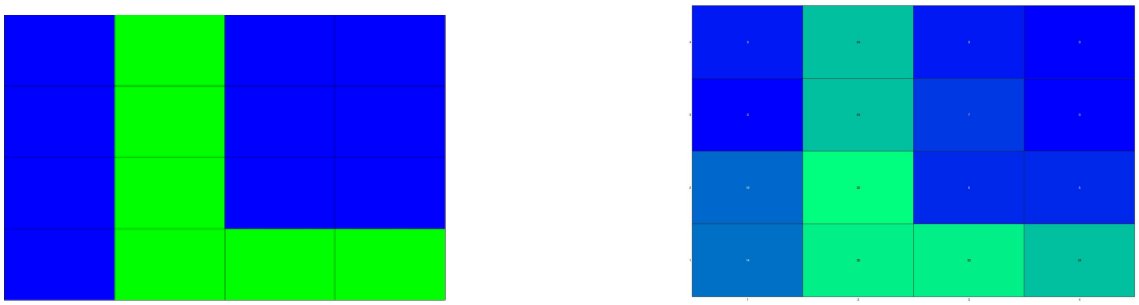

Figure 15: Test 1

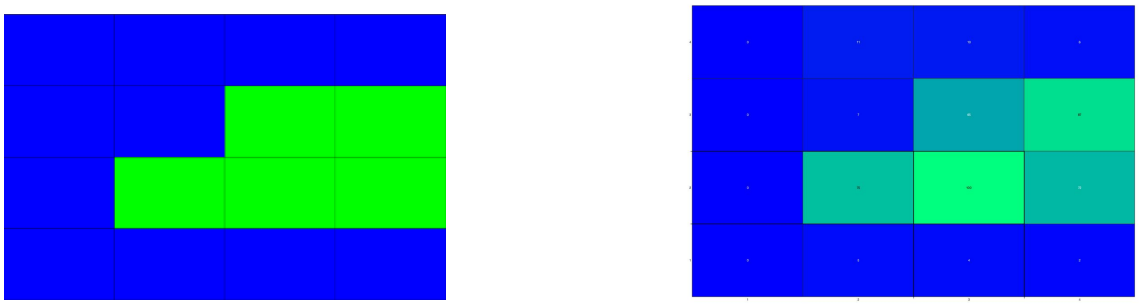

Figure 16: Test 2

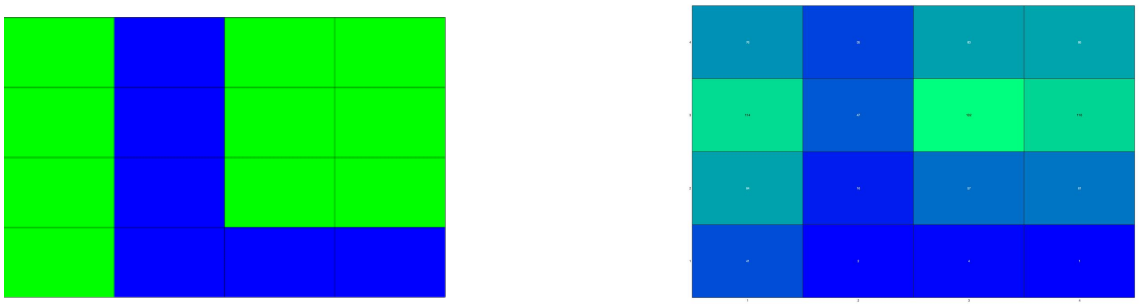

Figure 17: Test 3

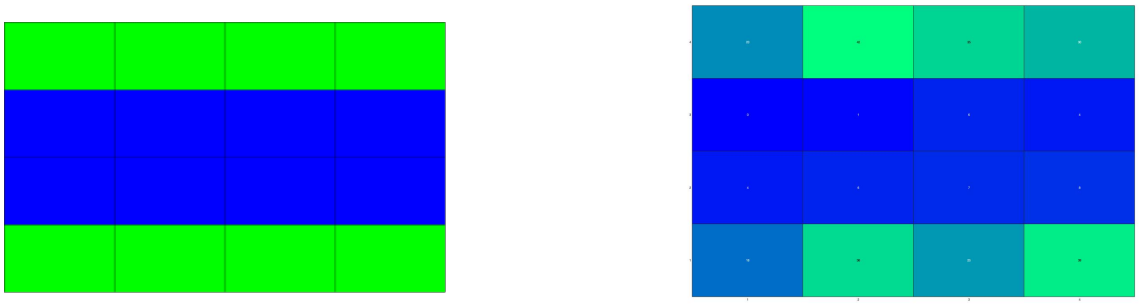

Figure 18: Test 4

8 Discussion and future works

Eye detection and gaze recognition is an interesting and challenging problem in computer vision area. It can be useful almost in every area of our lives. Even if a lot of different methods already proposed, this field is still developing, new “at-the-edge” technologies, applying to improve results of eye recognition.
The solution of the gaze prediction showed in this project works good for the shown case. But still, there can be some improvements, which allow to increase performances and accuracy of gaze prediction.
The explained method has significant benefit, because it only requires a web camera to perform gaze estimation.
The experimental results, also show that the proposed method can tolerate slight head movement, which is an important requirement, when integrating head pose information the acquisition with head pose-free gaze estimation. Moreover, the scaling factor allows to estimate the gaze even if the head distance changes.
By the way, the algorithm suffers from changing the light conditions. In particular case, when the sun points directly to the face, the detection algorithm didn’t performed as desired.
Moreover, our algorithm is strictly dependent from the calibration phase. In the test phase, if the model has not been fed with data from different distances, it won’t be able to track the gaze properly.
The target of the next task could be to explore head pose-free and calibration-free gaze estimation method.
In future work we propose to find a way to analyze the performances of this solution, in a mathematically and analytically way, since it was not always easy for us to understand how much a change was impacting the overall solution of the problem.
The rotation of the face is not cover by our approach. A solution can be explored using a rotation matrix.
Another important step that can be done to improve our solution, is to change the way to extract the features, with a supervised descent method (SDM) that could lead in a more accurate estimation.

References

[1] Ferhat, Onur Vilariño, Fernando. (2016). Low Cost Eye Tracking: The Current Panorama. Computational Intelligence and Neuroscience. 2016. 1-14. 10.1155/2016/8680541.

[2] Jones, Michael, and Paul Viola. "Fast multi-view face detection." Mitsubishi Electric Research Lab TR-20003-96 3.14 (2003): 2.

[3] [3] Lucas, Bruce D., and Takeo Kanade. "An iterative image registration technique with an application to stereo vision." (1981): 674.

[4] Skodras, Evangelos, Vasileios G. Kanas, and Nikolaos Fakotakis. "On visual gaze tracking based on a single low cost camera." Signal Processing: Image Communication 36 (2015): 29-42.

[5] Loy, Gareth, and Alexander Zelinsky. "Fast radial symmetry for detecting points of interest." IEEE Transactions on pattern analysis and machine intelligence 25.8 (2003): 959-973.

[6] https://www.peterkovesi.com/matlabfns

[7] Bour, L. J., M. Aramideh, and B. W. Ongerboer de Visser. "Neurophysiological aspects of eye and eyelid movements during blinking in humans." Journal of Neurophysiology 83.1 (2000): 166-176.

[8] Feng, Guo-Can, and Pong Chi Yuen. "Variance projection function and its application to eye detection for human face recognition." Pattern Recognition Letters 19.9 (1998): 899-906.

[9] Zhou, Zhi-Hua, and Xin Geng. "Projection functions for eye detection." Pattern recognition 37.5 (2004): 1049-1056.

[10] Shi, Jianbo. "Good features to track." 1994 Proceedings of IEEE conference on computer vision and pattern recognition. IEEE, 1994.

[11] Skodras, Evangelos, and Nikos Fakotakis. "Precise localization of eye centers in low resolution color images." Image and Vision Computing 36 (2015): 51-60.

[12] Cai H, Yu H, Zhou X and Liu H. "Robust gaze estimation via normalized iris center-eye corner vector." Intelligent Robotics and Applications: 9th International Conference, ICIRA 2016.