## 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.

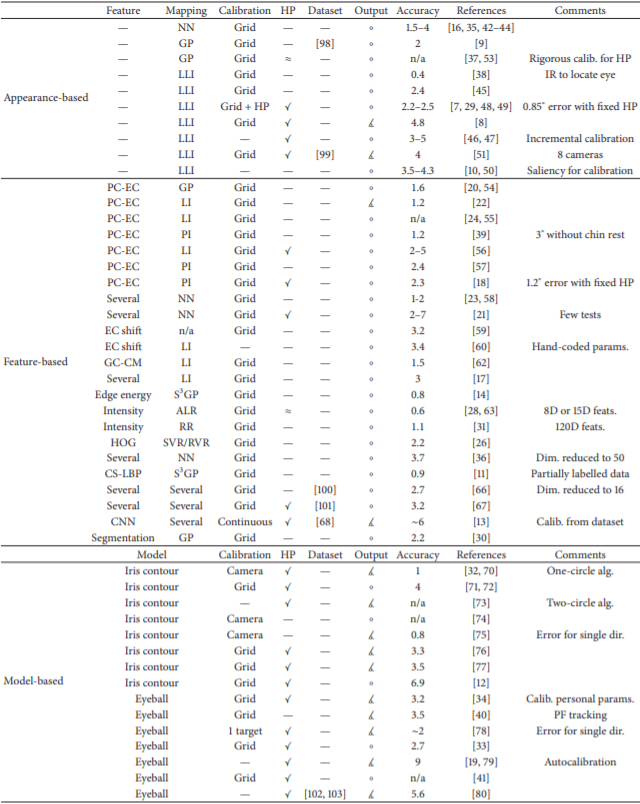
## State of the art

Devices or systems that track a person’s eye movements are called eye trackers or gaze trackers. Currently, the most widespread techniques used in these trackers make use of light sources and cameras that operate in the infrared (IR) spectrum. There are many available commercial models that are in the form of either eyeglasses or table mounted devices and also open source alternatives that allow the use of custom hardware.

Visible light gaze tracking, on the other hand, does not require any special hardware and aims to solve the task making use of regular cameras. In this paper, we will concentrate on this class of trackers and survey the related research.

With the aim of clearly identifying the borders between different visible light gaze estimation techniques, there are different categorization scheme:

* *Appearance-Based.* These methods only use the eye image pixel intensities to create a mapping to the gaze estimation. The image pixels are converted to a vector representation via raster scanning and fed to the estimation component.
* *Feature-Based*. Methods of this category also make use of a mapping to calculate the gaze; however, they use richer feature vectors compared to the methods in the previous category (i.e., not just pixel intensities).
* *Model-Based*. Compared to the discriminative approach of the first two categories, the methods belonging to this category follow a generative approach by trying to model the eyes and maybe even the face. The gaze is calculated geometrically using the model parameters.

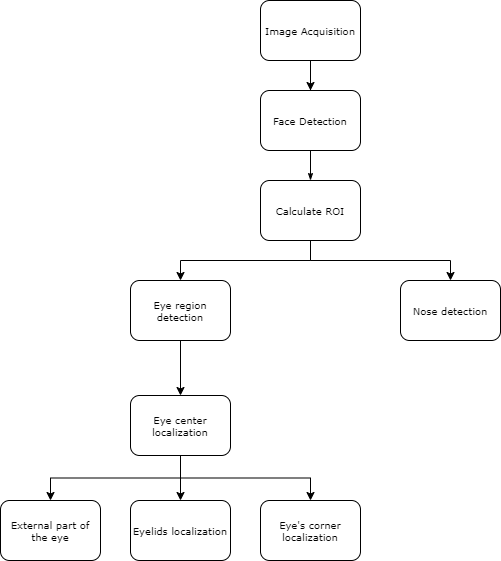
In the table below, summary and results of all the techniques analyzed in [1] are shown. Methods are grouped into categories for easier reference. HP column shows whether the technique has head pose invariance or not. Techniques allowing small head movements are denoted by ≈ symbol. Output column shows what type of gaze is calculated: point of gaze (∘) or line of gaze (∡).

## Code Flowchart

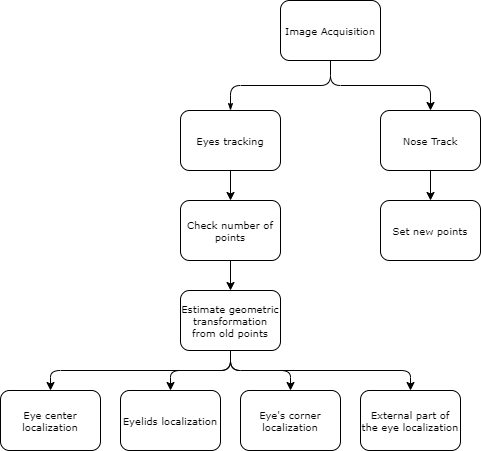
In this section we will describe the code from a general perspective, with the help of a flowchart. To make the model more precise, we decided to distinguish the program into two parts. In the first part the user can repeatedly run the calibration phase and in the second part, the user will be tested 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 prediction, and the calibration.

**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 is applied.

**The 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, eye center, eyelids, medial angle eye and external part of the eyes are calculated, and the tracking is initialised.

**The 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.

**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).

**Calibration:** the calibration procedure is done only in the calibration version. The calibration is needed to calculate the coefficients of the linear regression model. The calibration procedure will show a window with some blue points moving on the screen. When a point is showed look at it until the final `bip' is heard and the point move in the next position. The calibration pattern consists in 9 points placed in a square arrangement.

## 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 where to 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 you can see the eye ROIs in cyan boxes.

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].

**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 channel. The skin area has a red dominant resulting in higher values in the channel. Also, the separation of the luminance information makes the skin modelling more resilient to different lighting conditions and uneven illumination. Figure 4 shows a converted image of the left eye ROI.

**eyeMapC calculation** The is the result of the combination of and channels to highlight the eye area. The eyeMapC is defined as

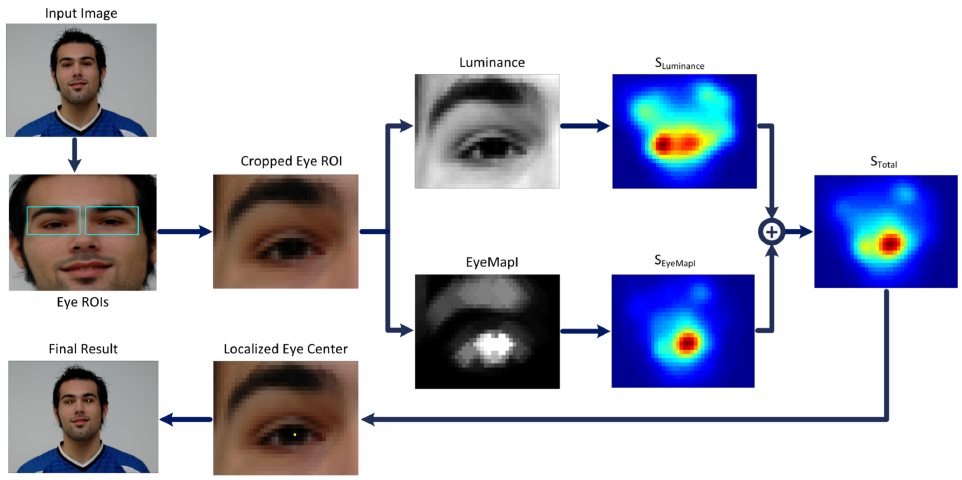
where , are normalized to the interval [0,1] and Cr means (1 - Cr). The , division could produce or pixel values. is saturated to the max numerical value and is replaced by 0. Large values on the eye map are observed at the positions of the eye regions and eyebrows where the colour difference from the skin pixels is maximized.

Figure 1 - Overview of the proposed system. For clarity purposes, the images showing radial symmetry transform results are pseudo-colored. Image taken from [11]

**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 with the luminance channel. Therefore, we calculate a new eye map , that is, the division of by , the luminance channel. The process is graphically represented in Figure 5. The has high values in the iris area while 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 component and the brighter appearance in the component. We perform these operations with two at circular structuring elements called B1 and B2, for and respectively. The radius of these circular elements is defined with respect to the iris radius.

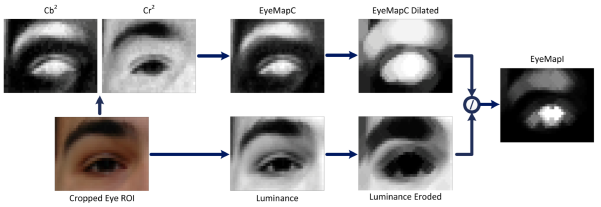
where and denote gray-scale dilation and erosion, respectively. is used to solve numerical problems deriving from a zero division.

Figure 2 - 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 is used:

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 𝑌 𝐸𝑟𝑜𝑑𝑒𝑑 and to the 𝐸𝑦𝑒𝑀𝑎𝑝𝐼. Then, the two transforms , are summed together and the position of the maximum value pixel is the position of the eye center

**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 build one patch for each eye. These two patches are located near to the inner sides of the eyes, the ones between the eye’s corners and the nose. A patch contains the inner eye corner and the eyebrow edge, therefore comprising a highly textured area containing edges which are easy to track robustly. The dimensions of the patches depend on the interocular distance. The interocular distance is simply the distance between the two detected eye center.

**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 0.1 ∗ and 0.3 ∗ correspondingly ( stands for the interocular distance); each vertical side is at a distance of 0.05 ∗ 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 0.15 ∗ 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 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

where is the rectangle center (eye center), 𝜌 = 0,1, … , 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.

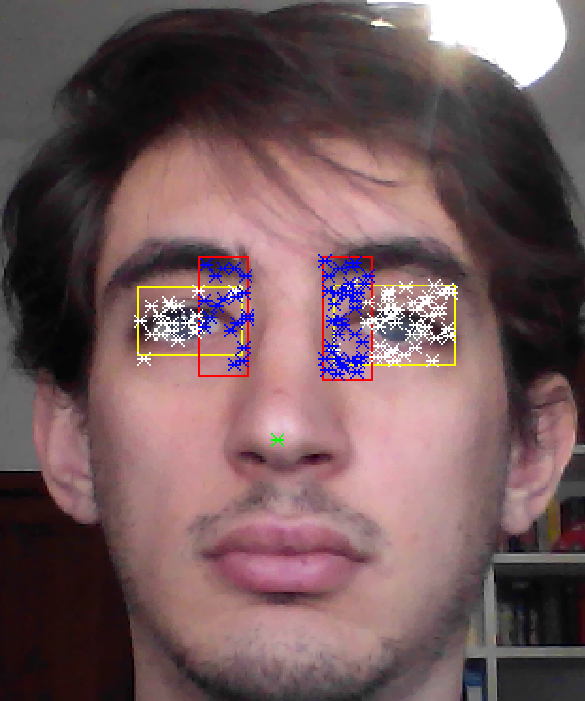
Given the search ROI for the eyelids, denoted here after as , we first perform an edge detection on the cropped image, through canny filter. The integral projection function of Eq. (3) is computed for with ϑ being the inclination of the line connecting the two detected eye centers, which represents the 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.

**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.

## Tracking

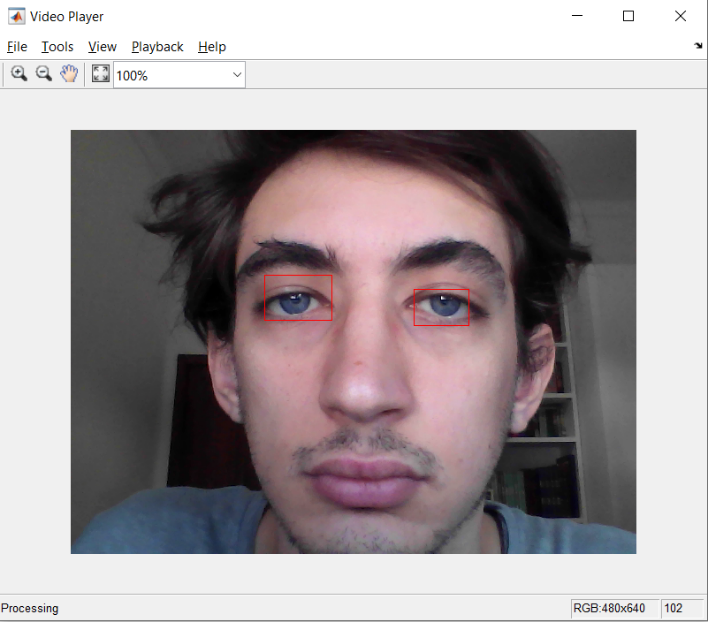
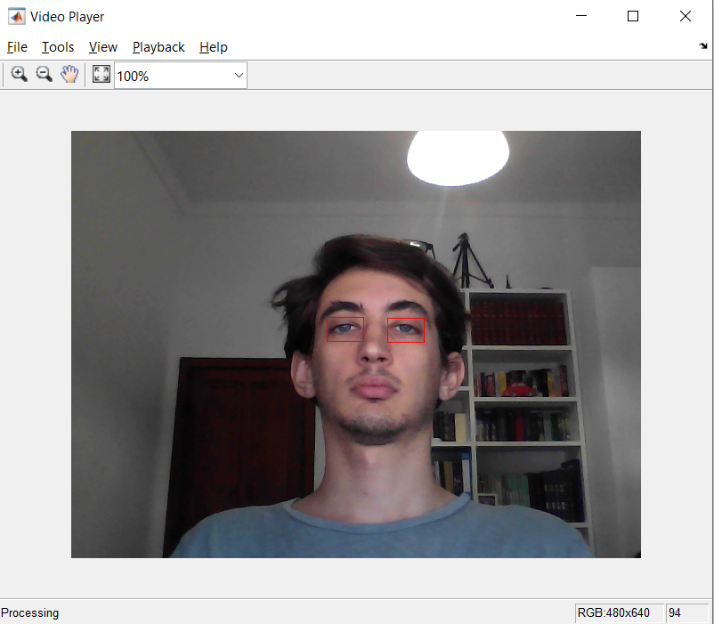
Tracking is the process of locating a moving object or multiple objects over time in a video stream. Tracking an object is not the same as object detection. Object detection is the process of locating an object of interest in a single frame. Tracking associates detections of an object across multiple 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 tracker is initialized using the entire frame image and as ROI the patch previously found. 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.



## 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. The prediction model is the piece of code that map image data to screen positions, that is, the gaze direction. 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.

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 distance between two points is collected, is than scaled to the bbox found in detection and tracking phase. 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.

To have a fast calibration procedure and for the reasons above explained, a linear regression approach is used. Four mapping functions, one for each eye direction (along x and y axes), are learned.

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.

Then, as mentioned before, the horizontal and vertical features are respectively scaled by the two factors and

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.

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

* Horizontal features:
  + Horizontal distance between eye center and nose tip
  + Horizontal distance between eye center and eye corner
  + Horizontal distance between eye corner and nose tip
  + Horizontal distance between the external part of one eye to the other
* Vertical features:
  + Vertical distance between eye center and nose tip
  + Vertical distance between eye corner and nose tip
  + Vertical distance between upper eyelid and eye center

## Calibration

In the calibration phase 9 points are shown to the user, by lighting them one by one and registering the features vectors in the known positions. After calibration there are correspondences between known points on the screen and their relative features, allowing the train the linear regression model. This operation is performed by the script “gen\_model.m”.

## Test

As explained, when the calibration phase is completed, the new input vector arrives, and 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 divided in green and blue boxes, the user is asked to look at the green ones. After this phase is finished, an heatmap will appear, showing the user, if he was able to recreate the pattern.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

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, one by one.

## Discussion

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 is more robust against slight head movement, which is an important requirement, when integrating head pose information the acquisition with head pose-free gaze estimation. Future work will be targeted to explore head pose-free and calibration-free gaze estimation method.

In future work we propose to find a way to analyse the performances of our 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.

## References

[1] Onur Ferhat, Fernando Vilariño. "Low Cost Eye Tracking: The Current Panorama" Computational Intelligence and Neuroscience, February 2016.

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

[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.