## Introduction

Eye - gaze tracking is one of the most challenging problems in computer vision. In this project we faced the challenge to improve the accuracy and the consistency of this task that was previously face up from other students with Matlab.

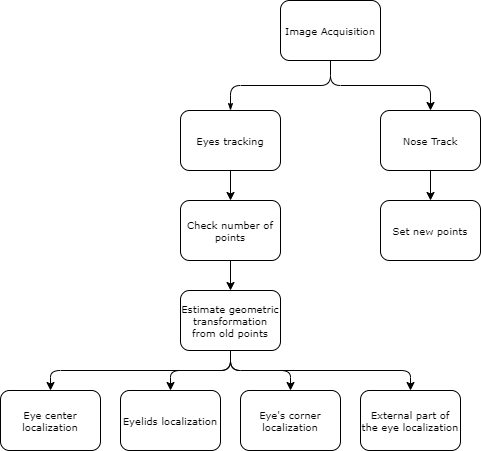
The goal of this project was to show how gaze tracking can provide assistance in the act of detect where the user is looking on a screen for monitoring how much is he focusing on ad in a web page. It can find application in the commercial world and advertisement. To do so, it was needed to overcome some challenges, like detection of face and eye features in the videoframe, their tracking in real time and the estimation of the point of gaze from the features.

Gaze Tracker is a Matlab program that localizes the eye center and plots the estimated gaze on the screen using a Feature-Based approach. The program is thought to work in real-time with a single low-cost camera, such those you can find on PCs, tablets, and smartphones.

## State of the art

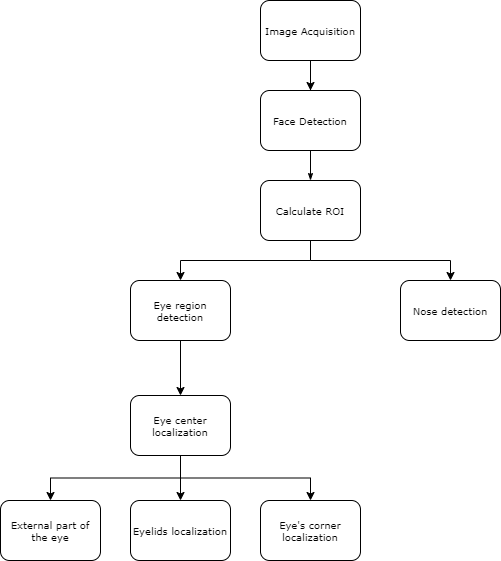
Several approaches and techniques have been proposed in the literature. For a complete overview of the state of the art please refer to the paper “On visual gaze tracking based on a single low-cost camera” [1].

## 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 in two part where in the first part you can repeatedly run the calibration phase and the second part in which the user will be tested with a random table to test the accuracy of the gaze estimation.

This first part can be divided in six section, thar are, 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* 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, 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. 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.

**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.

**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. 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) 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 I'm 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 procedure is furtherly explained in [5]. 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. 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 [4]. 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.

Another problem that we found is that, since the features are collected with a certain distance from the screen, the gap between the is learned based on that; 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. Two mapping functions, one for each direction (along x and y axes), are learned. Given the assumption of independence of gaze estimation in the two axes, two separate feature vectors are formed as horizontal and vertical distances between moving and anchor points.

Features for both axes are calculated as the following:

* Horizontal features:
  + Horizontal distance between left eye center and nose tip
  + Horizontal distance between left eye center and left eye corner
  + Horizontal distance between left eye corner and nose tip
  + Horizontal distance between right eye center and nose tip
  + Horizontal distance between right eye center and right eye corner
  + Horizontal distance between right eye corner and nose tip
  + Horizontal difference between the width of the left bbox and the width of the right bbox
* Vertical features:
  + Vertical distance between left eye center and nose tip
  + Vertical distance between left eye corner and nose tip
  + Vertical distance between left upper eyelid and left eye center
  + Vertical distance between right eye center and nose tip
  + Vertical distance between right eye corner and nose tip
  + Vertical distance between right upper eyelid and right eye center

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

## Calibration

In the calibration phase some points, randomly chosen, are showed to the user, lighting them one by one and registering the feature vectors in that known positions. After calibration I have correspondences between known points on the screen and the relative features, allowing me to train the linear regression model. This operation is performed by the script “gen\_model.m”.

## Test

As explained, with the calibration completed, the new input vector arrives, and it is mapped to the estimated screen point with the obtained model parameters. To approximate a real situation where we want to know how precise the estimation is, we setup multiple grids in which is asked to the user to look at the green boxes. After this phase is finished, an heatmap will show the result and if the user were able to recreate the pattern that was showed to him.