Computer Vision
Gaze estimation

Nicola Gastaldello (1206748)

A.Y. 2019-20

Abstract

The purpose of this experience was to make automatic gaze estimation. Using different images, through C++ with the `OpenCV` library, I tried to classify the gaze direction into three categories, i.e., looking straight, looking right or looking left.

1 Introduction

Gaze estimation is a very useful technique because it can improve the lives of users with disabilities enabling their interaction with computers. In recent years, these technologies have become widely used in both research and commercial applications.
However, this task is still being studied due to the presence of several challenging difficulties. In particular, the main problems deal with the low quality of sensors or unknown/challenging environments, the large variations in eye region appearance and in images and videos with lower number of pixels, motion blur, less ideal illumination conditions and ambiguity in determining the extent of the eyeball or shape of the iris.

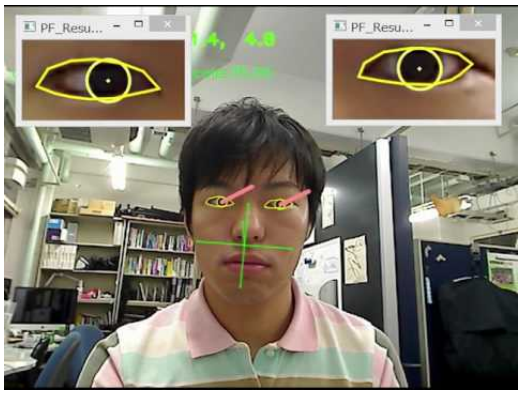

Figure 1: General example of gaze estimation.

My approach to solve the gaze estimation problem is based on colors, i.e., in one eye there will be a relatively dark colored component composed by iris and pupil and a white component which is the sclera. The solution is provided by a histogram which detects the relative position of both iris and sclera inside the eye in consideration. More specifically, as shown in figure below, the eye of a person who is looking straight has a central colored part and two external white parts, whereas a person who is looking left

has colored region in the left part of the eye and white region in central and right part, and vice-versa for a person who is looking right.

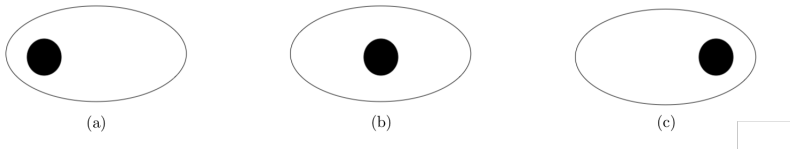

Figure 2: (a) Looking left; (b) Looking straight; (c) Looking right.

2 Procedure description

To develop my solution of gaze estimation problem I used C++ and the `OpenCV` library. The algorithm is divided in six steps: image blurring, face and eye detection, image thresholding, edge detection, image cropping and rotatation and finally the histogram computation. The following sections offers a brief analysis of all the steps.

2.1 Image blurring

After uploading the image to be examined, i blurred it. Image smoothing is a very useful image processing operation to capture significant patterns in the data, while leaving out noise or other fine-scale structures/rapid phenomena. The adopted filter is the Gaussian one, through the `cv::GaussianBlurr`. Gaussian filtering is done by convolving each point in the input array with a Gaussian kernel (size 3 x 3) and then summing all them in order to generate the output array.

Figure 3: On the left the uploaded image, on the right the blurred image.

2.2 Face and eye detection

To detect the face and the eyes I used the `cv::CascadeClassifier` method. Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Viola and Jones. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

I uploaded the pretrained models and then I performed detection using `cv::detectMultiScale` method, which returns boundary rectangles for the detected faces or eyes, as shown in figure below. It works by finding all the faces in an image, and then finding the eyes inside every detected face.

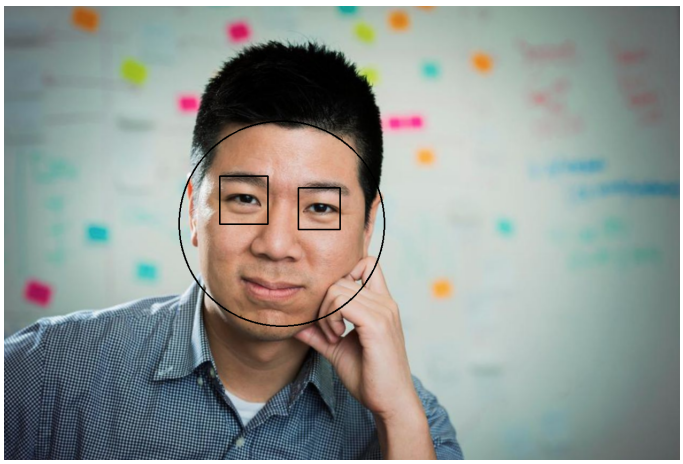

Figure 4: Output result of Haar-Cascade classifier.

2.3 Threesholding

After the detection, I decided to select only one eye for each face. So, after the converstion into grayscale, I created a new region of interest which considers only the detected eye and I applied a threshold. I used this tecnhique because it represents the simplest segmentation method and so more distinction between colors can be achieved, namely, between dark region (iris and pupil) and light region (sclera). In particular, in the algorithm, a binary threshold was applied via `cv::threshold`, i.e., above a certain intensity value of the pixel, it assume the black value, otherwise white.

Figure 5: Output result after threshold the region of interest.

2.4 Edge detection

For a more accurate analysis I also applied edge detection, because with Viola e Jones method we found a region which contains other elements besides the eye. So, in order to select strictly only the eye portion,

Canny edge detection was used, because it is the optimal detector whose aims are: (1) low error rate; (2) good localization; (3) minimal response. The `cv::Canny OpenCV` function was applied by setting different values of low threshold and kernel size. Finally, I created an ellipse using the longer edge detected.

Figure 6: Detection of eye with Canny edge detector.

2.5 Crop and rotate

The final eye portion is the region inside the ellipse created with Canny detection. I cropped this subregion of the image and, using the `cv::warpAffine` function, I applied an affine transformation to rotate it. This because some images can contain person with tilted head and for the final analysis the orientation is important.

Figure 7: Output result after crop and rotate the region of interest.

2.6 Histogram computation

Once created the image to analyze, as mentioned before, I adopt a histogram approach to classify the direction. This is done by the usage of a local pixels operation, namely, I made the average of the intensities of all pixels in one column. After this step, I created a histogram which contains on the y-axis the average of the pixel intensities and on the x-axis the respective column, in order to maintain the relative position as in the image.

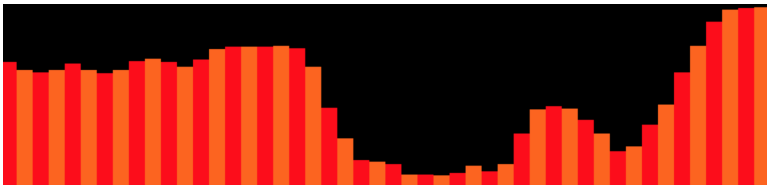

Figure 8: Output histogram of the average intensities.

3 Results

Finally, to determine the gaze direction we have to analyze the histogram (example in figure above), namely, low intensity value means color region (iris and pupil) and high intensity value means white part (sclera). I used two techniques:

- Manually estimation, i.e., looking at the histogram we observe the position of the region with lower intensities values (i.e., left, center or right).
- Automatic estimation, i.e., divide the histogram in three equal components and compute the average for each one and check if the lower average value is in the left, in the center or in the right.

Figure 9: New hist with 3 bins for automatic estimation.

With this results we can distinguish if a person is looking left (low intensities values in the left), looking straight (low intensities values in the center) and looking right (low intensities values in the right). Manually estimation works better then the automatic estimation, but setting the parameters of all the functions used, this approach works quite good in almost all test images. In the figure below are reported some results.

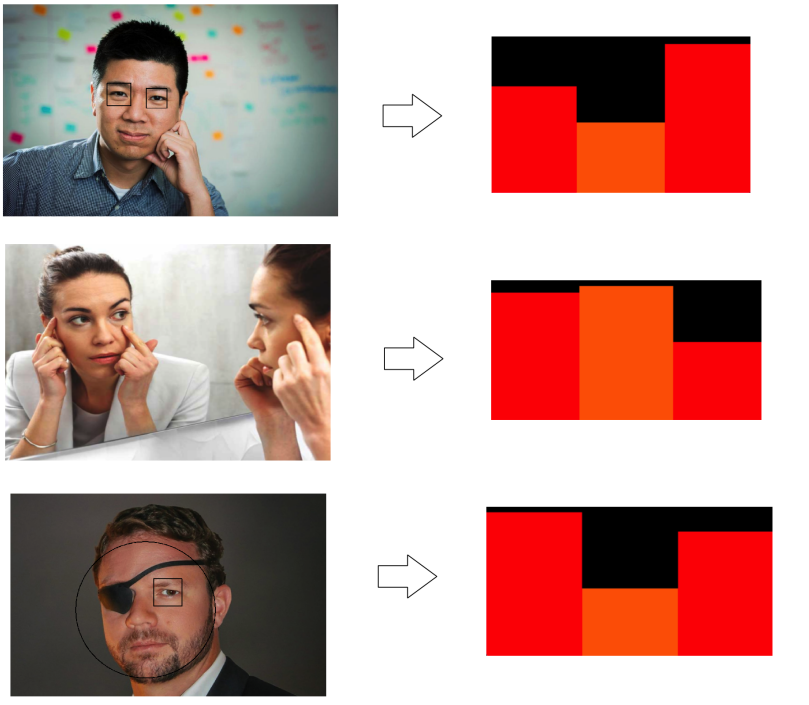

Figure 10: Three results with automatic gaze estimation.

4 Optimization

Analysing the results, I found that the three main problems are:

- **Image quality:** low resolution, strange illumination and distance of the person from the camera affect the whole process. One example can be found in some images with a lot of people, because subjects are far away and eye regions become very small. Another example could be in images with portion of the eye very pixilated and thus we have poor number of pixel to make analisys.
- **Face and eye detection:** Sometimes Cascade classifier does not work well and it does not recognize face or eye regions.
- **Edge detection:** Due to the incorrect threshold or to spurios elements, Canny edge detection is not able to create the border for the eye region.

Therefore, to optimize this algorithm it may be helpful to remove more noise, improve face or eye detection and use a different technique to select only the eye. Another useful task to be implemented is the automation of the choice of the best parameters for each method.
After these considerations, since the results in terms of direction classification are quite good, this approach can be considered efficient to perform automatic gaze estimation.

Figure 11: Example of low resolution image which leads to a difficult gaze estimation.

Figure 12: Example of spurios elements after thresholding.

Figure 13: Example of incorrect Canny edge detection.