Report: Gaze Eye Estimation

Aurora Zabot, 1206742

ICT for Internet and Multimedia — A.Y. 2018/2019

Introduction

The main aim of this project is to develop a procedure in order to localize eyes in the proposed images, grasp their color and detect the gaze direction. For sake of clarity, this report will be ordered in three main sections, in which I'll analyze the different methodologies used to the aforementioned problems. In each portion I'll also add ideas relative to the possible other implementation and comments about the difficulties faced to realize each part.

1 Algorithm

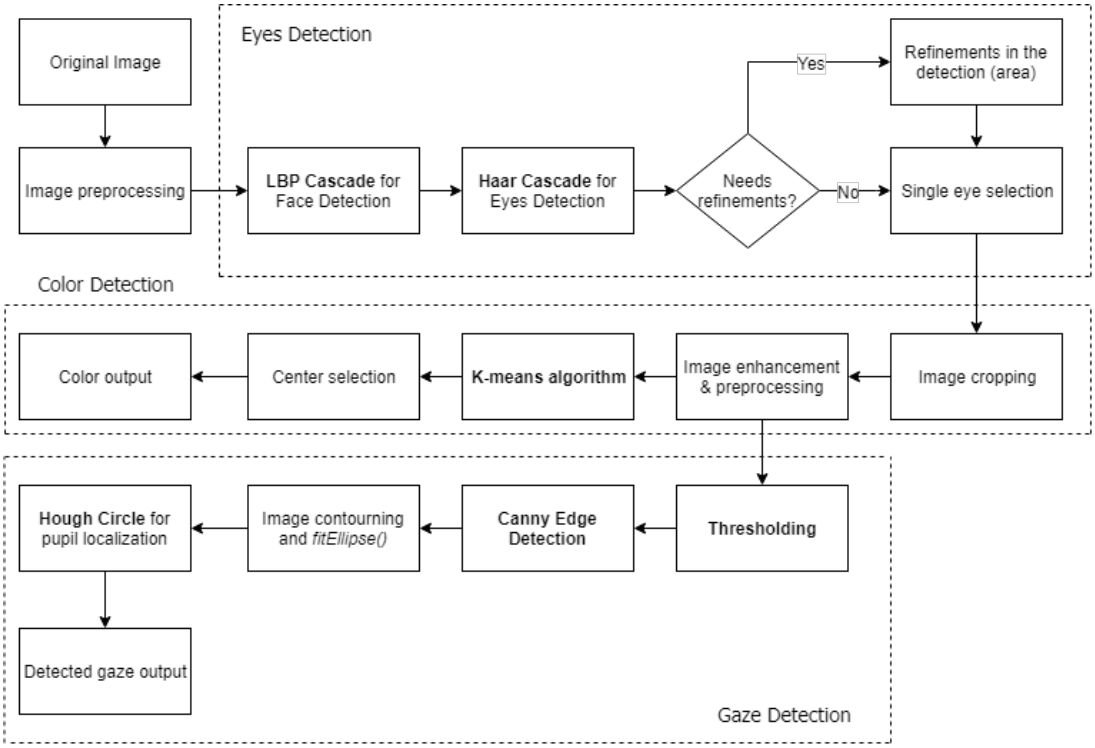


Figure 1: Algorithm implementation scheme

2 Implementation

In this implementation I've combined several algorithms and tools in order to satisfy the various requests of the project. The first paragraph (2.1) is just an introduction of the side methods used to enhance the image and manipulate the various parameters.
For the very first part, the *Eye Localization* (2.2), I used the **Viola and Jones** method, implementing two

cascades: the first to find faces inside the image (using an **LBP cascade**), whereas the second one to find eyes (through **Haar cascade**). The reasons to make this distinction would be better explained in the appropriate section.
In the second section, the *Color Estimation* (2.3), I've implemented the **K-means** algorithm in order to grasp the dominant color of the iris portion of the image, previously enhanced by a conversion through *HSV color space*.
The last part regard the implementation of the *Gaze Detection* (2.4); here I just processed the image using the **Canny Edge** algorithm and the **Hough Circle** transform in order to find the pupil and find the gaze orientation.

2.1 Preprocessing

In this section I would expose the kind of side methods I've used in the implementation and the reasons behind the choice of the parameters in use. In order to do so, I'll summarize everything in a table to be schematic, since this is just an overview to better understand the following paragraphs.

Method	**Description**
intensityEqualization	Compute the equalization of the intensity (graylevels) through a color space conversion using the method *cvtColor* and *equalizeHist*; this method would be used in the 2.2 part.
luminanceAdjustment	As the method above, it equalizes the *luminance* component taking advantage of the *YCbCr* color space; it's used in the 2.2 portion as well.
saturationEnhancement	The method passes through the *HSV* color space conversion, enhancing just the saturation channel (and not equalizing it); it will be used in the 2.3 part.
hsvEqualization	Equalization of the *saturation* and *brightness* channel through a color space conversion from *BGR* to *HSV*. It has been used in the 2.4 part.
imageBlur	Through the *kind* input we can select if we would use a *Gaussian* or a *Median* kernel in order to blur the image (the second preserves edges, while the first smooth them as well). Then, if we select the *Gaussian* one, with *selection* it can be decided if the output image would be blur (with a low pass) or sharp (with an high pass, using a mask to invert the filter). This method has been used in every portion of the code to preprocess the image.
edgeDetection	Implementation of the **Canny Edge Detection** algorithm, that is quite slow than other implementations, but more reliable. The two thresholds have been selected with a rate of 2:1, since from the documentation this is the best choice to achieve good results [1]. The higher one has been chosen quite high in order to detect only very strong components. This method has been used in the 2.4 portion.
morphOperation	Implementation of the morphological operation of *closure*, using a circular kernel of size 3x3. It can be used in the 2.4 part, but the results on the set of images I'm working on are worse using it.
houghCircle	Used in the 2.4 part to find pupils. We convert the image from *BGR* to *GRAY* in order to input it the **Hough Circle** method. The *accumulator* parameter has been selected considering that is related to the accuracy of the detector. If it's too high, we would detect a lot of false negative, but instead a too low choice can compromise the actual detection of the pupil. To make a trade between those two conditions, I've chosen a 2.0 accumulator value. The *minimum value* parameter has been taken quite high in order to avoid double selections [2]. The *high threshold* from the documentation should be the same used in **Canny** [3]. The choice of the *radius* has been made considering that the pupils have a small circumference.

2.1.1 Optimization

- For the **Canny** implementation I could use **Otsu's method** to calculate the two thresholds to achieve the best results in output [4]

2.2 Eye localization

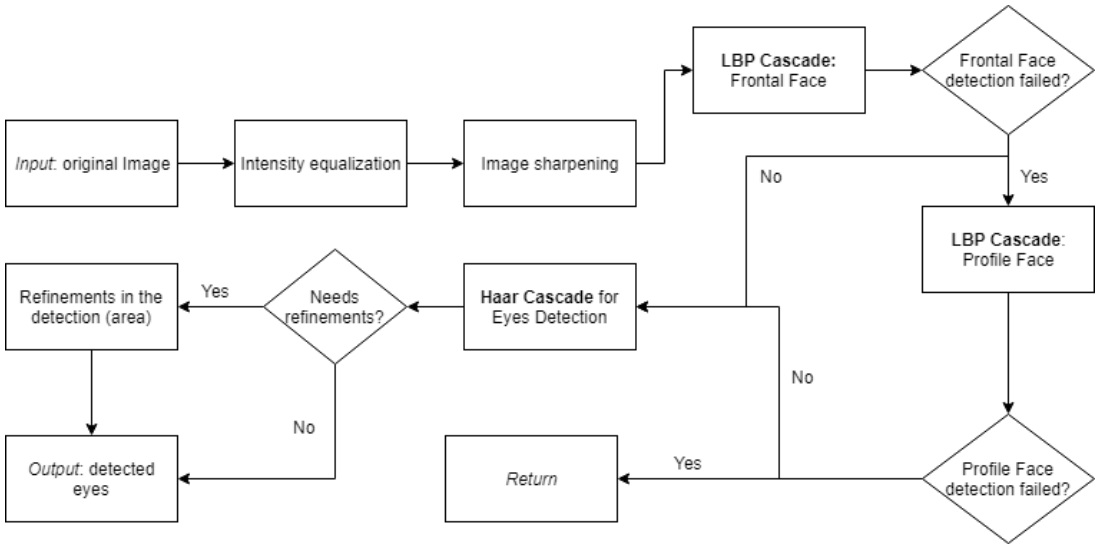


Figure 2: Eye detection implementation scheme

For localize eyes in the images proposed I'd used the scheme reported in figure (2). Following the **Viola and Jones** approach, I've implemented two cascade of classifiers. The parameters in use in both the cascades have been selected considering that a good *scaleFactor* value could be taken as 1.05, in order to reduce the size of 5% to obtain a better detection, and a *minNeighbors* value that is suggested to be around 3-6 to achieve good performances [5].
The first cascade using the **LBP** approach has been employed to detect frontal and profile faces; this choice has been made considering that if we don't detect them at the first step, the entire implementation fails, since trying to find eyes in the whole image would return a lot of false positives (besides the high complexity). In this sense, the **LBP cascade** is useful to lighten the computational complexity and detect faces easily. It may have more false positive than the **Haar** one actually, but most of the time with the second step we resolve the problem, since in those areas no eyes are find.
The second classifier uses a **Haar cascade**, since is more reliable than the **LBP**. In this case, I'm trying

(a) Eye detection before refinements (b) Eye detection after refinements

Figure 3: Comparison before/after refinements

to localize eyes in the faces previously found. Most of them are actually find correctly, but there are in some cases mismatches, in particular with nostrils and mouth contours. In order to solve this problem, I've considered the differences in the areas: if the detection output more than one element, compute the area

of the ROI and select just the two components with the highest and similar area. If there are no matches, select just the two major areas (but also in this case with a certain margin in the difference of areas), since statistically we would have that nostrils have smaller area than eyes in the detection. Instead, if the detection find just one element, select it without additional computations, since it could be we're dealing with blindfolded faces. The differences before and after the refinements can be seen in figure (3a) and (3b).
In the output I've just decided to pass the component with the major area (if the detection correctly finds the pair of eyes) in order to lighten the computation; this choice has been made in particular for images with multiple people, since in that case the computational complexity resulted quite high.

2.2.1 Observations

(a) Eye detection without preprocessing (b) Eye detection on B&W image

Figure 5: Eye detection with sharp image

In order to achieve good results in the eye detection we need to preprocess a bit the image. As we can see in figure (4a), if we take the input as it is, we would obtain a lot of mismatches even though we pass through the refinement part of the code. Indeed, also passing the B&W image in the **Haar cascade** the results are quite poor, as we can see in image (4b), while we reach a good point using it in the **LBP cascade** [10].
I've reached the best result sharpening the image a bit though an *Gaussian* high pass filter, as it can be seen in figure (5).

2.2.2 Problems

The main problems faced during this part of the computation are related to three kinds of elements: eyeglasses, blindfolds and when there are many people. In particular, combining two or more of those elements, it becomes really difficult to recognize faces and, in particular, eyes. A solution to this kind of problem may be found in the *optimization* subsection (2.2.3).

2.2.3 Optimization

- In order to achieve better performances, we can train manually the cascades, inserting *negative* images to be sure that the detection will be more consistent [6].

2.3 Color estimation

In this portion of the implementation I've though to a sort of segmentation approach to find the color of the eye, that can be summarize by the following scheme (6).

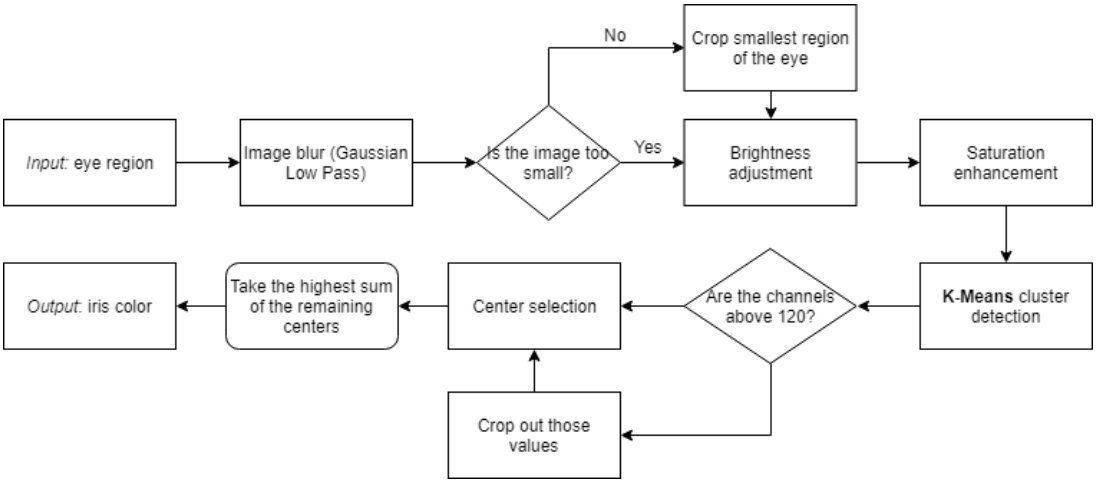


Figure 6: Color detection implementation scheme

At the very beginning of the computation, I've preprocessed the image, blurring it with a *Gaussian* low pass filter, in order to attenuate noise effects and smooth edges; then I cropped the image if the size weren't too small in order to cut off the biggest portion of the skin and focus on the eye region. To do so of course I need to rely a lot on the right computation of the first part of the implementation.
In order to perform a good color segmentation, I've enhanced the brightness and the saturation components through the two methods aforementioned in the *Preprocessing* 2.1 paragraph. The saturation enhancement in particular is important to underline the pure *BGR* components, needed to the correct detection of the iris color.
After this preparatory part I've run the **K-Means** algorithm in order to find color clusters. I've chosen the following parameters [7]:

- *Number of clusters*: 5, since I thought that in this way we've found the ruddiness, the iris, the pupil, the white of the eye and another spurious component as color clusters
- *Number of attempts*: 5, in order not to make the computation heavier
- *Flag*: I've used the *KMEANS_PP_CENTERS*; from the documentation is the best way to find clusters in real data
- *Term Criteria*: I've chosen to stop at the maximum number of iterations or if I've reached a certain level of accuracy

In order not to add weight, I didn't output the cluster image, but only the rounded vector of centers.
From this point on I've worked on centers, cropping from the vector all the darkest components and taking the highest peak (considering the sum of each channel). This, would be, with high probability, the color of the iris, recognized as the dominant component in the vector. The color is roughly returned in output, since I've just make a differentiation from brown, green and blue [9].

2.3.1 Observations

Since the iris and the pupil are circular shaped, in the first version of the implementation I've searched to manipulate the image in order to apply blob segmentation, at the beginning with **SimpleBlobDetector** and then with **MSER**. In both cases the attempt fails due to oversegmentation. Probably initializing inner markers we would avoid this kind of problem [8].

2.3.2 Problems

The main problems in this case can be summarize in two elements: with a single person, the detection fails if the eyes are quite closed, since the dominant color grasped would probably be the skin one. This is the case of the image (7), that in the set is the only wrong detected of this kind (the eye is categorize as *brown* instead of *blue*); on the other hand, if images have an high number of people it could be that the

Figure 7: Wrong color detection

classification doesn't go well, since the eyes are too small respect to the background. The other problem is inside the algorithm itself: **K-Means** is a really light and fast method, but it's too much linked to the initial choice of the clusters. If we initialize them wrongly due to the randomness, the computation would surely fail.

2.3.3 Optimization

- In order to achieve better results we should use a more reliable segmentation approach (maybe **Mean Shift** could be a good idea)

2.4 Gaze detection

The last portion of the implementation is related to the *Gaze Detection* and it's summarize in the scheme (8) below.

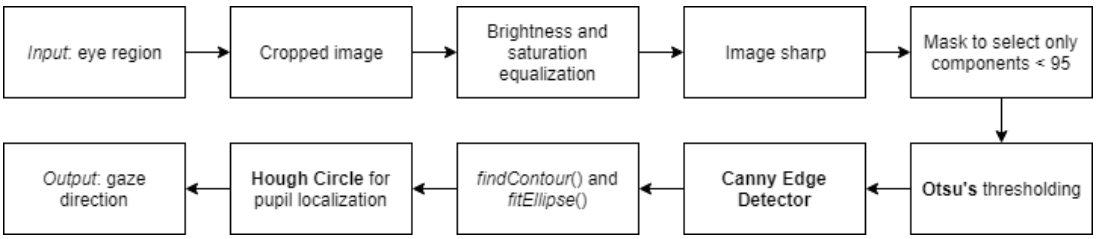
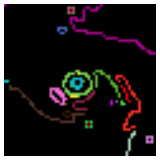


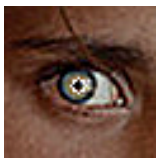
Figure 8: Gaze detection implementation scheme

The main aim of the computation is to find the pupil in order to grasp its relative location inside the eye. In order to do so, I needed to manipulate a bit the images, in particular using **Otsu's thresholding** and the **Canny Edge Detector**. Before those ones, I just equalize the brightness and saturation of the images through a color conversion from *BGR* to *HSV* using one of the aforementioned methods implemented in (2.1). Then, using another of the firsts method as well, I sharped the image through a *Gaussian* high pass. One of the main steps of this portion of the implementation is the application of the mask to save just the darkest components, in particular with a range of $[0, 95]$. I've tried many other specifications in this sense, also using a range slightly lower, but in my set of images I've obtained the best results with those values. Then I apply **Otsu's method** to obtain a better thresholding on the image, taking also in this case a really small range, saving just the components in the range $[245, 255]$. Then I used **Canny Edge Detector** to find

(a) Original Image



(b) Contour Image



(c) Pupil detected

Figure 9: Pupil detection steps

edges; I've chosen this method since from the theory we know is the most accurate one. It may add a little bit of computational complexity, but since we need an exact location of the pupil in the eye we can tolerate it to achieve better performances.

After this, I've implemented a method to find contours of the image and fit ellipses in it. I've made a discrimination in the ellipses just saving the ones with the two similar lengths (since I'm looking shapes that are almost circles) and that have a certain minimum size [11]. I applied **Hough Circles Transform** (see chapter (2.1)) on the resulted image in order to locate the pupil. One example of the resulting computation can be seen in figure (9).

The last step of the implementation considers where the pupil is located inside the eye region and output the gaze direction, considering that if the subject is looking left or right, it would has the pupil drastically moved on one side of the eye.

2.4.1 Observations

In the middle passage from mask selection to edge detection, at the very beginning of the implementation, I've also used *morphological operators* in order to improve my results, in particular the *closing* operation. Theoretically, this operation should "close" weak and spurious edges. But in this case, since the images are quite small due to the fact that I crop just the eye portion, probably it not works so well and a lot of elements merging together, increasing substantially the amount of mismatches.

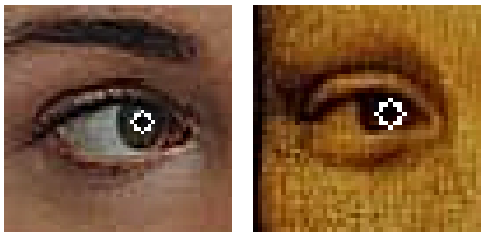
2.4.2 Problems

In this case, the problems faced can be divided into two categories:

- Correct detection of the pupil but wrong gaze classification (see figures (10b) and (10a))
- Wrong detection of the pupil location (if the classification is correct anyway by chance, I assume it is wrong), see figure (11)

For the first point, the main problem is related to the right-look classification. In fact, most of the eye images in my dataset are, by chance, left eyes; this means that we have a more "long" component on the right side, for how the eye shape is. In this sense, we should consider a larger area for the right detection in the last step, but in this way we may misclassify more general images. So I prefer maintain this condition unaltered.

For the second point, in my dataset I've found just a serious wrong classification; probably also in this case is related to the fact that the eye region is pretty small.



(a) Right mismatch (b) Portrait mismatch

Figure 10: Right pupil detection, wrong classification: both classify as looking straight

Figure 11: Wrong detection of the pupil: classify as looking straight

3 Results

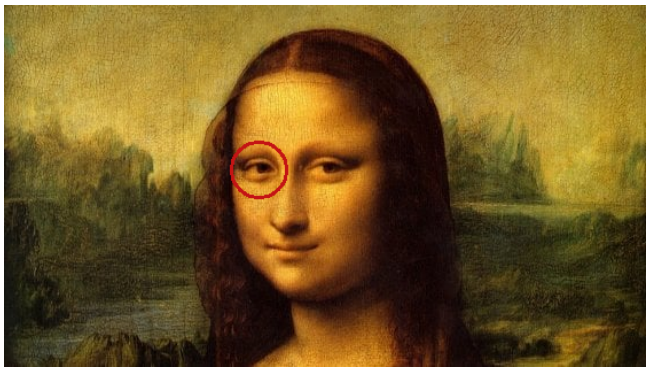
In this section I would take a look on the main results of my computation. I'll discuss on the main issues related to the dataset in use and which images make more mismatches for each section.

3.1 Dataset and mismatches

3.1.1 Eye localization

As we can clearly see, the main issues are related to multiple people images and to portraits. This is something that would return also in the following sections.

Dataset	Mismatch	Comment
	Doesn't detect the left subject and one eye of the right subject	This image is quite noisy, probably the mismatch is due to this fact

	Double detection on one eye	Refinements fail, with high probability
	Doesn't detect the right eye	The main problem is probably related to the fact that the image is quite dark

3.1.2 Color estimation

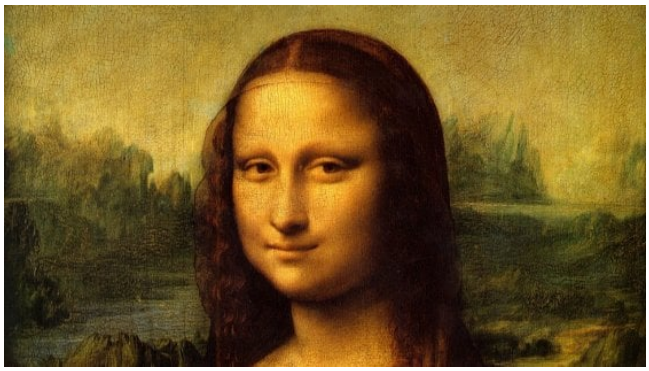
As I've already told before, the main problem in this section is when eyes are quite close, so the main component may be detected as the ruddiness one (so the red one, marked as brown).

Dataset	Mismatch	Comment
	Detect brown eyes	Since the eye is quite close, probably the dominant color detected is the red due to the ruddiness

	Two mismatches, both output as brown instead of blue	Probably the problem is related to the dimension of eyes; indeed, since K-means compute randomly centroids, sometimes this problem is overcome

3.1.3 Gaze detection

Here I've faced a lot of issues in particular with multiple people images and with right-looking faces.

Dataset	Mismatch	Comment
	Output: look straight	As I've aforementioned, with the right eye detection I've faced more problems, probably related to the fact that we're analyzing a left-eye
	Output: look straight	Same problem as the image above

	Output: look straight	As I've already said, in this case fails the pupil detection
	A pair of eyes is not matched looking right	In this case the main problem is the quality of the image, that doesn't allow a good classification of the gaze detection

4 Conclusions

As we can see from the images and the analysis proposed, we can see that in particular portraits and multiple people images gives more mismatches than the single person images. Moreover, the main problem with the gaze detection is given by the right look if the portion of the eye taken in analysis is the left one. One probable way to deal with this problem is to take both the couple of eyes in the initial computation and make a match in the orientation (but, on example, in the *Game of Thrones* cast image this may fail in any case, since one actress cross eyes), adding of course a bit of computational complexity. Another approach may be using *Machine Learning* algorithms and *Deep Learning* approaches in order to make the analysis more refined (and probably speed up the computation, since besides the train of the Neural Network, this kind of computation normally is enough fast).

References

[1] OpenCV documentation,
`https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html?highlight=canny%20edge`

[2] Stack Exchange,
`https://dsp.stackexchange.com/questions/22648/in-opecv-function-hough-circles-how-does-parameter`

[3] OpenCV documentation,
https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=houghcircles#houghcircles

[4] StackOverflow,
https://stackoverflow.com/questions/4292249/automatic-calculation-of-low-and-high-thresholds-for

[5] StackOverflow,
https://stackoverflow.com/questions/20801015/recommended-values-for-opencv-detectmultiscale-para

[6] OpenCV documentation,
https://docs.opencv.org/trunk/dc/d88/tutorial_traincascade.html

[7] AI Shack,
http://aishack.in/tutorials/kmeans-clustering-opencv/

[8] Learn OpenCV,
https://www.learnopencv.com/blob-detection-using-opencv-python-c/

[9] buZZrobot,
https://buzzrobot.com/dominant-colors-in-an-image-using-k-means-clustering-3c7af4622036

[10] Wikipedia,
https://en.wikipedia.org/wiki/Unsharp_masking#Digital_unsharp_masking

[11] OpenCV documentation,
https://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/bounding_rotated_ellipses/bounding_rotated_ellipses.html