Assignment #01

IT University of Copenhagen (ITU)
Introduction to Image Analysis and Machine Learning
(Spring 2017)

March 8, 2017

Purpose In this assignment, you will develop eye tracking methods to detect eye feature like pupil, glints and iris, as shown in Figure 1. One common strategy for performing eye tracking is first to detect the pupil and glints, and then the iris. Pupil and glints centers are in simple cases often determined through thresholds and connected component labeling. On the other hand, iris contour can be detected by using pupil position as a starting point.

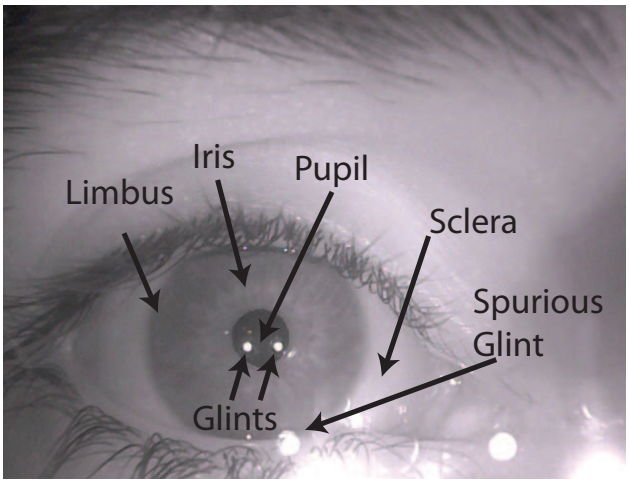

Figure 1: Specific eye feature used by eye tracking methods.

You will also develop a classifier to select the best threshold for a skin color detector method. This assignment does not have one grand-solution, in fact, there is plenty of room to make individual implementations and testing out your ideas.

Deadline This assignment spans three weeks: *Week #01*, develop methods for pupil and glints detection; *Week #02*, develop an iris detection method and use mathematical morphologies to improve the pupil/glints detector; *Week #03*, develop a skin color detector and improvements for your eye tracker. You will have time during the exercises sessions to implement the techniques but it is expected that you spend more time outside class. Your report and developed code should be submitted on March 29, 2017 at 13:00 through learnIT.

Groups This assignment must be handed in groups of 2-3 students. You can NOT mix bachelor and master students in a group, because there are exercises with bachelor and master levels. As with any other group work, it is important that you distribute the workload among the members. Notice that we will expect more work to be done by master students groups than by bachelor students groups (in term of performance and robustness of your code). As soon as possible, inform Fabricio (fabn[at]itu[dot]dk) or Zaheer (zahm[at]itu[dot]dk) about the names of your group member. Your group ID will be posted on learnIT.

Report The reports will be evaluated between the groups by a peer-review system (peergrade.io) and by the teachers. We will evaluate your results based on the text as well as the recorded videos. If you won't pass, you will have more 2 weeks (after receiving our comments) to correct your report. The resubmission deadline will be informed on learnIT. Make sure your report, Python project, source code and output videos follow these instructions:

- The first page of your report must contain your **names** and **emails** in the front page;
- You must delete the folder `Inputs` before zipping your final project;
- The ZIP filename should be as follow: *Assignment1_GROUPINDEX.zip*;
- A link where the output videos can be download (if your `Outputs` folder is really big);
- Your report must contain only the image sequences that you have made yourselves (i.e. your own dataset); and
- The Python scripts that you have changed (as an *Annex*).

Equipments During the experiments, you will use a web camera to make your own dataset. Borrow the cameras from either Fabricio or Zaheer in Office 4D25. Notice: You can use any night vision cameras and infrared light sources (850 nm wavelength) to get a better light conditions and clear eye feature. For some cameras, it is possible to cover the light sensors so the cameras use IR light. The web camera you will borrow has a switch to turn on/off the infrared light sources.

Input Data Each group will evaluate their methods using at least the videos files associated with their Group ID (available on learnIT) according to the Table 1. However, it is important to also evaluate other video sequences to ensure that you have a robust implementation of your detection methods. These videos are available in `Input` folder.

Table 1: Videos to be processed by each group.

Group ID	Filenames
01 and 11	eye01.m4v, eye11.m4v and your own dataset
02 and 12	eye02.m4v, eye12.m4v and your own dataset
03 and 13	eye03.m4v, eye13.m4v and your own dataset

Continues on the next page.

Table 1 – continued from previous page.

Group ID	Filenames
04 and 14	eye04.m4v, eye14.m4v and your own dataset
05 and 15	eye05.m4v, eye15.m4v and your own dataset
06 and 16	eye06.m4v, eye16.m4v and your own dataset
07 and 17	eye07.m4v, eye17.m4v and your own dataset
08 and 18	eye08.m4v, eye18.m4v and your own dataset
09 and 19	eye09.m4v, eye19.m4v and your own dataset
10 and 20	eye10.m4v, eye20.m4v and your own dataset
	End of Table 1

Your Results The results of your experiments will need to be saved and shared with your report. In each exercise, you will find a section called *Output* with the filenames that you must save of your output videos.

Files included in this assignment The supporting material of Assignment #01 contains the following folders, packages and source code:

⋆ indicates files you must complete
∓ mandatory for master students, optional for bachelor students
† indicates optional exercises

`CaptureVideo` – Package for controlling multiple input video sources (videos or images)
`Inputs` – Folder with some input video examples (*.m4v)
`Outputs` – Folder with your processed output videos
`RecordVideo` – Package for controlling the video recording process
`ex1.py` – Python script that steps you through the Exercise 1.01 (Pupil Detection)
`ex2.py` – Python script that steps you through the Exercise 1.02 (Glints Detection)
(†) `ex3.py` – Python script that steps you through the Exercise 1.03 (Morphologies)
`ex4.py` – Python script that steps you through the Exercise 1.04 (Iris Detection)
(∓) `ex5.py` – Python script that steps you through the Exercise 1.05 (Improvements)
`ex6.py` – Python script that steps you through the Exercise 1.06 (Skin Color Detection)
(⋆) `EyeFeatureDetector.py` – Class to detect eye feature from images
`GeometricMethods.py` – Python file with methods of geometric transformations
(†) `RegionProps.py` – Class to get descriptors of contour-based connected components
(⋆) `SkinColorDetector.py` – Class to detect automatically the skin color from images
`warmUpCapture.py` – Simple example script to capture videos from a webcam or a file
`warmUpRecord.py` – Simple example script to record output videos

Week #01

Exercise 1.01. *Pupil Detection* (2.5 hours) – You will develop a method to detect the pupil and calculate its center. The pupil and iris are darker than their surroundings. For this reason, you will use image sequences obtained using IR light to ensure good contrast among the iris and the pupil.

For the report In the report, you must specify your assumptions on how to detect pupils. Show the detected pupil center in the original image sequence, e.g. in a series of not necessarily consecutive images. In the report you have to discuss under which circumstances your method fails/works and why?

Tasks

(a) You will develop a code to identify all possible pupil candidates in the input image. Pupils can be represented as an ellipse with 5 parameters (Figure 2) (x_0, y_0, a, b, θ), i.e. the center (x_0, y_0), the longest axis a, the shortest axis b and the angle θ between the x-axis and a. Change the method `getPupil()` in `EyeFeatureDetector.py` file, so that it returns a set of pupil candidates (i.e. `ellipse` parameter) and their center coordinates (i.e. `centers` parameter). The method `getPupil()` already find all pupil candidates using the function `cv2.findContours()`[1].

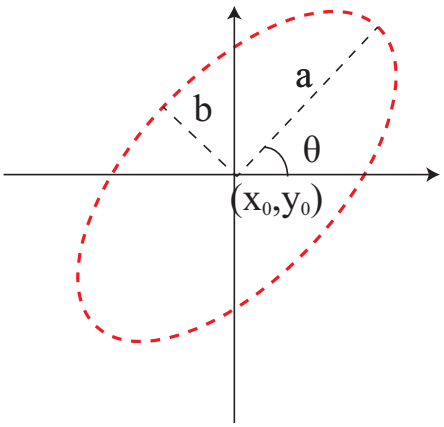

Figure 2: Parameters of an ellipse.

(i) Extend the `getPupil()` method so it iterates though each blob and calculate the region properties using `calcContourProperties()` (defined in `RegionProps` class).

(ii) Use the command `ellipse = cv2.fitEllipse(contour)`[2] [more than 4 points] or `ellipse = cv2.minAreaRect(contour)`[3] [less than 5 points] to fit an ellipse to the contour points (i.e. the current pupil candidates).

(iii) Update the `getPupil()` method so that it calculate the center of mass of an ellipse. The center is obtained through `Centroid` property in `calcContourProperties()`

[1] An OpenCV function to find contours in a binary image.
[2] An OpenCV function to fit an ellipse around a set of 2D points.
[3] An OpenCV function to find a rotated rectangle of the minimum area enclosing the input 2D point set.

method. Probably, your implementation may be a bit slow (don't worry it will be taken care of soon).

(iv) Add the current ellipse in the `ellipses` vector and the current ellipse center in the `centers` vector. Use the method `append()` for that.

(v) Evaluate your current method using the input video files according to the Table 1.

□ **Output:** Record three videos with the results of your current pupil detector. You should name these files as `Ex101_A01.mp4`, `Ex101_A02.mp4` and `Ex101_A03.mp4` .

(b) As you see, your current method returns a lot of false positive pupil candidates. Develop a filter for connected components (a.k.a. blobs) based on regions properties to remove most of these pupil candidates. Region properties can be used to specify classification rules used to determine which blob corresponds to the pupil. A good start is using the size of the contours to determine which blobs correspond to the pupil.

(i) Create classification rules (e.g. `if Area > a and Area < b or ...`) to select the blobs that corresponds to good pupil candidates using only on the `Area` property of the blobs. The pupil size varies, but only within certain limits. *Tips:* Area values between 500-6000 could be a good starting point.

(ii) Check all properties of `RegionProps` class to create your classification rule.

(iii) Adapt your classification rule to include `Extend` from `calcContourProperties()` method (in addition to "`Area`") to improve the results.

(iv) Change the method `getPupil()` to only insert the pupil candidates that fulfill your classification rule, instead of inserting all pupil candidates in the vectors `ellipses` and `centers`. Don't worry if it is not perfect yet. You probably have to run your program several times until you find good pupil candidates.

(v) Make sure that `ex1.py` script shows the original image, the processed binary image (thresholded image), and the centers of all detected pupil candidates.

(vi) Evaluate your current method using the input video files according to the Table 1.

□ **Output:** Record three videos with the results of your current pupil detector. You should name these files as `Ex101_B01.mp4`, `Ex101_B02.mp4` and `Ex101_B03.mp4`. *Hint:* You must use the same input videos that you have used in Exercise 1.01 (a).

(c) Now your current pupil detector returns only the pupil candidates filtered by your classification rule. However, your code can still identify some false positive pupil candidates. Develop a classifier to choose only the true positive pupil candidate.

(i) Make the method `getPupil()` returns the index of the true positive pupil candidate (i.e. `bestPupil` parameter) in the vectors `ellipses` and `centers`.

□ **Output:** Record three videos with the results of your pupil detector. You should name these files as `Ex101_C01.mp4`, `Ex101_C02.mp4` and `Ex101_C03.mp4`. *Hint:* You must use the same videos that you have used in Exercise 1.01 (a) and (b).

(d) Plot the distribution of the data based on two properties of the pupil candidates (e.g. Area *vs.* Extend). Select different pupil properties and check how to apply one supervised classifier to select the best pupil candidade. In the method `getPupil()`, before returning the results, call the method `__plotData()` to see the pupil distribution. *OBS*: Please, change the name of `__plotRegressionData()` method to `__plotData()`.

(e) *(Optional)* You can define more properties in the `RegionProps` class. Include the "Circularity" feature in the `RegionProps` class to check how circular a blob is. It can help you to choose the best pupil candidate because the pupil boundary is very close to a circle. Feel free to add new features according to your needs.

(i) Create a method called `__CalcCircularity()` that given the contour of a blob (`points`), returns the blob circularity based on the Equation 1:

$$circularity = \frac{4 \times \pi \times Area}{Perimeter^2} \tag{1}$$

(ii) Change the method `calcContourProperties()` to update the dictionary `props` when the "Circularity" property is passed in the `properties` argument. Remember to call the method `__CalcCircularity()` and include the returned value in `props`.

(iii) Use the circularity feature to choose the best pupil candidate in the method `getPupil()`.

(iv) Discuss in your report which are the circularity values that give pretty good results for your pupil detection method.

(v) Evaluate your current method using the input video files according to the Table 1.

□ **Output:** Record three videos with the results of your improved pupil detector. You should name these files as `Ex101_E01_Optional.mp4`, `Ex101_E02_Optional.mp4` and `Ex101_E03_Optional.mp4`. *Hint:* You must use the same videos that you have used in Exercise 1.01 (a), (b) and (c).

Exercise 1.02. *Glints Detection* (1.5 hours) – You will develop a method to detect glints (i.e. corneal reflection) and calculate their centers. Glints can be found using a similar strategy as you have used in pupil detection. Your method will detect the true glints candidates while reducing the number of *false positives*.

For the report In the report, you should specify your assumptions on how you detect glints. Show the detected glints centers in the original image sequence, e.g. in a series of not necessarily consecutive images. In the report you have to discuss under which circumstances your method fails/works and why?

Tasks

(a) The method `getGlints()` defined in `EyeFeatureDetector.py`, but it has not been implemented. Develop the method `getGlints()` so that given an image and a set of parameters (a threshold, the minimum and maximum glint area, and the number of glints) it will return a set of glints candidates (i.e. `ellipse` parameter) and their center coordinates (i.e. `centers` parameters).

(i) Use thresholding, blob detection and blob filtering, just like for pupil detection, to get the reliable glints candidates.

(ii) Start with just using `Area` property in `calcContourProperties()` for the classification. You should expect to detect several spurious glints.

(b) Based on the connected components and some rules for how the glints and pupil change appearance over time, you should be able to detect the center of the pupil and glints in most of the frames in the image sequences.

- **Output:** Record three videos with the results of your glints detector. You should name these files as `Ex102_B01.mp4`, `Ex102_B02.mp4` and `Ex102_B03.mp4`. *Hint:* You must use the same videos that you have used in Exercise 1.01 (a), (b) and (c).

(c) The method `getGlints()` contains a argument called `numOfGlints`. Make the method `getGlints()` fills out the vector `bestGlints` and returns the indexes of the N best glints candidates based on the number of glints informed by the user.

- **Output:** Record three videos with the results of your glint detector. You should name these files as `Ex102_C01.mp4`, `Ex102_C02.mp4` and `Ex102_C03.mp4`. *Hint:* You must use the same videos that you have used in Exercise 1.01 (a), (b) and (c).

Sometimes, it can be difficult to set parameter values (e.g. *threshold*) only with the blob (shape) properties of the individual feature. You will most likely have detected too many or too few blobs candidates in the sequences. At this stage, it is better to have too many pupil and glints candidates as some of these can be removed based on their distances to other eye feature. In the following steps, you will use the spatial relations between pupil and glints to get more robust results. For this purpose, you will make changes (implementation) in the method `getGlints()` to filter the glints candidates based on their distance to the pupil center (i.e. `pupilCenter` argument).

Recall that the (Euclidean) distance between two point is:

$$distance = \sqrt{d_x^2 + d_y^2},$$

where d_x and d_y are the differences between the x and y coordinates of the two points.

(d) Implement the method `__Distance()` in `EyeFeatureDetector.py` to calculate the Euclidean distance between two pair of point (i.e. `p1` and `p2`). You can use Numpy or SciPy to do this, e.g. the Euclidean distance between `u` and `v` is:

```
numpy.linalg.norm(u - v) # or
scipy.spatial.distance.euclidean(u, v)
```

(e) Create a classification rule that uses the distance between the pupil and glints, to remove those glints candidates that are too far away from the pupil center. This classification must be created in the method `getGlints()`. The pupil center coordinate of the best pupil candidate is passed in the argument `pupilCenter`.

- □ **Output:** Record three videos with the results of your glint detector. You should name these files as `Ex102_E01.mp4`, `Ex102_E02.mp4` and `Ex102_E03.mp4`. *Hint:* You must use the same videos that you have used in Exercise 1.01 (a), (b) and (c).

(f) *(Optional)* Are there other constrains that can be applied to make the results even better? Implement your ideas and evaluate your method.

(g) Evaluate your current method using the input video files according to the Table 1.

- □ **Output:** Record three videos with the results of your improved glint detector. You should name these files as `Ex102_G01_Optional.mp4`, `Ex102_G02_Optional.mp4` and `Ex102_G03_Optional.mp4`. *Hint:* You must use the same videos that you have used in Exercise 1.01 (a), (b) and (c).

Week #02

Exercise 1.03. *Mathematical Morphology* (1 hour) – You will use mathematical morphology (i.e. erosion, dilation, opening and closing) to improve your eye feature detection methods. For example, it can remove some *false positive* candidates and filling out holes.

For the report In the report, you should discuss the differences in your methods before and after using the mathematical morphology operators. Discuss which morphological operators would be best suited if you want to avoid the pupil pixels to merge with iris pixels? Discuss which morphological operators would be best suited if you want to remove the reflections in the pupil area?

Tasks

(a) Improve the method `getPupil()` in `EyeFeatureDetector.py` so that it uses mathematical morphology on the thresholded image to achieve better results.

- □ **Output:** Record three videos with the results of your pupil detector. You should name these files as `Ex103_A01.mp4`, `Ex103_A02.mp4` and `Ex103_A03.mp4`. *Hint:* You must use the same videos that you have used in Exercise 1.01 (a), (b) and (c).

(b) Improve the method `getGlint()` in `EyeFeatureDetector.py` so that it uses mathematical morphology on the thresholded image to give better detection results.

- □ **Output:** Record three videos with the results of your glints detector. You should name these files as `Ex103_B01.mp4`, `Ex103_B02.mp4` and `Ex103_C03.mp4`. *Hint:* You must use the same videos that you have used in Exercise 1.01 (a), (b) and (c).

(c) (*Optional*) Make additional trackbars that you can change your parameters interactively for pupil and glints detection. You might by now have many trackbars to manipulate in the OpenCV windows. However, this is just an eye tracker prototype. :)

Exercise 1.04. *Iris/Limbus Detection* (3 hour) – You will develop methods that detect the limbus (i.e. iris boundary). Limbus detection may be more challenging than detecting the pupil and glints detections. Before detecting the iris, your code should detect the pupil and use its center for searching the iris boundary.

For the report In the report, you should specify your assumptions on how you detect iris. Show the detected iris boundary in the original image sequence. In the report you have to evaluate your method (challenges, robustness, etc), discuss the challenges and advantages of the method, and under which circumstances your method fails/works and why?

Tasks

(a) Your iris detector will use gradient direction to detect the limbus and fit an ellipse around the iris. Develop the method `__GetGradientInfo()` in `EyeFeatureDetector.py` that given an image it will return the image gradient (G_x, G_y), the gradient magnitude (G_m) and the gradient orientation (G_o). Recall that:

- Orientation: $\theta(i,j) = \arctan(G_x(i,j), G_y(i,j)) * 180/\pi$
- Gradient magnitude: $\nabla I(i,j) = \sqrt{G_x(i,j)^2 + G_y(i,j)^2}$

Hint: You might want to use `numpy.gradient()`[4] or `cv2.Sobel()`[5] functions for making the gradient image.

(b) Describe, in your report, the properties of gradients, gradient magnitude and gradient orientation in the eye images.

Through the previous exercises (specially Exercise 1.01), you have hopefully found a good localization of the pupil and a rough estimate of its center and size. You can use this information to make a contour-based localization of the iris by first detecting edges and then fit an ellipse to a subset of the found edge pixels.

When using contour-based methods you can use the gradient magnitude calculated in the whole image, but you can also save computations by only calculating the gradients along 1D directions. In this exercise you will experiment with both method. Common for these methods is to measure edges in the image along the curve normals.

The function `getCircleSamples()` in `GeometricMethods.py` receives the center (`C`), radius (`R`) and the number of curve point samples (`N`), and returns a list of `numOfPoints` sample points on the circle (x, y) and the curve gradients (d_x, d_y). The following Python code shows how you can obtain curve samples from a circle using the function `getCircleSamples()`.

```
n = 20
c = (100, 100)
r = 40
P = getCircleSamples(center=c, radius=r, numOfPoints=n)

```

[4] A Numpy function to return the gradient of an N-dimensional array.

[5] An OpenCV function to calculate the first, second, third, or mixed image derivatives using an extended Sobel operator.

```
for (center, dx, dy) in P:
    # < Define normal length as max distance away from initial circle >

    # < Get the endpoints of the normal p1 -> p2 >

    # < maxPoint = self.__FindMaxGradientValueOnNormal(magnitude,
      orientation, p1, p2) >

    # < store maxPoint in points vector >

# < Fit points to model using least squares - cv2.fitEllipse(points) >
```

(c) Develop the method `__FindMaxGradientValueOnNormal()` in `EyeFeatureDetector.py` and display the points of maximum value in a temporary image for verification. Its pseudo-code is given bellow:

```
def __FindMaxGradientValueOnNormal(self, magnitude,
                                   orientation, p1, p2):
    # < Get int coordinates on the straight line between p1 and p2 >
    points = getLineCoordinates(p1, p2)
    normalVals = magnitude[points[:, 1], points[:, 0]]
    # < Find index of max value in normalVals >

    # < Return coordinate of max value in image coordinates >
```

(d) Implement the method `getIris()` in `EyeFeatureDetector.py` that given the input image, the detected pupil center and the pupil radius finds the iris boundary using the maximum gradient magnitude along the normals.

(e) Use the (curve) gradient direction to disregard those pixel gradients directions that are not sufficiently aligned with the curve gradient (e.g. the angle between them is small), as shown in Figure 3. For example, you can use the angle between the curve normal and image gradient. Notice the vectors should be normalized to $length = 1$.

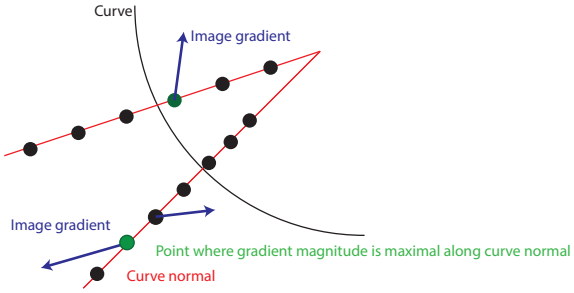

Figure 3: Curve normals (*red*), image gradients (*blue*) and feature point/point

□ **Output:** Record three videos with the results of your iris detector. You should name these files as `Ex104_E01.mp4`, `Ex104_E02.mp4` and `Ex104_E03.mp4`. *Hint:* You must use the same videos that you used in Exercise 1.01 (a), (b) and (c).

Week #03

Exercise 1.05. *Ideas and Questions for Improving Your Eye Tracker* (1-3 hours) – You will improve your current pupil, glints and iris detectors (in terms of speed and accuracy). This exercise is mandatory, however, it has different aims to bachelor and master students. Groups of *bachelor students* will select at least one of those topics and *only discuss* the improvements in the report. On the other hand, *groups of master* students will select at least one of those topics, *develop* the improvements and *discuss* them in their report. Choose as many of these as you have time for and not necessarily in the following order. Please, don't hesitate to contact us if you want some guidelines for how to proceed with these improvements.

For the report In the report, you should discuss the improvements you have chosen (and master students should present the improvements that you have developed for your eye tracking system). Show some pictures to illustrate the improvements achieved during this task. You have to discuss what are the differences in the eye feature detection methods before and after using the improvements.

- What effect does histogram equalization have on the eye images? Can it be used to improve or simplify your eye feature detection method? (*Difficulty: Simple*)
- Use Laplacian to detect glints (corneal reflections). (*Difficulty: Simple*)
- Develop a method called `getAutoThreshold()` in `EyeFeatureDetector.py` that can identify the best threshold for pupil and glints detection based on histogram analysis. Change the exercises scripts to use this method. (*Difficulty: Simple*)
- The eye corners are ocular feature defined over a small region that may be detected easily by the template matching technique. Develop a method called `getEyeCorners()` in `EyeFeatureDetector.py` that use template matching for eye corners detection. (*Difficulty: Simple*)
- The pupil can be used to restrict the search for the eye corners, i.e. the eye corners are always on the opposite sides of the pupil. You can extend the method `getEyeCorners()` so that it is able use the pupil position to constrain the search for the optimal location of the eye corner templates. (*Difficulty: Simple*)
- Experiment with several sequences to be able to document the properties of using template matching for eye corner detection. (*Difficulty: Simple*)
- Using image pyramids for pupil detection and iris detection to achieve a better speed in your eye feature detector. (*Difficulty: Medium*)
- Tracking can be done by storing the previous state (e.g. position and size) and use that information in the following frame to detect the eye feature in a sub region of the analyzed image (ROI – Region of Interest). *Tips*: use the previously found locations to filter those eye feature candidates that are too far away. (*Difficulty: Simple*)
- You can use the eye corners and iris circumference to detect the sclera. Implement a method that does this. Could we have used the sclera to detect the other feature? (*Difficulty: Medium*)

Only master students must record videos with the results of their improvements.

- ☐ **Output:** Record three videos with the results of your eye feature detector. You should name these files as `Ex105_A01.mp4`, `Ex105_A02.mp4` and `Ex105_A03.mp4`. *Hint:* You must use the same videos that you have used in Exercise 1.01 (a), (b) and (c).

Exercise 1.06. *Skin Color Detection* (2 hours) – You will work with analysis of histograms to select automatically the best threshold (using face recognition) for a skin color detector, as shown in Figure 4. You will use Haar features-based cascade classifiers to detect the user's face and then, analyze the histogram of the detected region. OpenCV provides the training data as XML file, and these files are available in `Inputs` folder. Feel free to create another approach to select the threshold of skin color. However, remember to discuss your approach in the report. In the end, as an *optional* exercise, you will extend your code to select the threshold for others object detection approaches.

Figure 4: Example of skin color detection with auto thresholding.

For the report In the report, you must specify your assumptions on how to detect the skin color. Show the developed skin color detector using images and videos captured from a web camera. In the report you have to discuss under which circumstances your method fails/works and why?

Tasks

(a) Implement the method `getSkinColor()` in `SkinColorDetector.py` file, that given an input image it will return the processed image with the detected human skin.

(b) Use the function `cv2.CascadeClassifier.detectMultiScale()`[6] to detect faces in the input image. All detected faces will be returned as a list of rectangles, in which (x, y) means the upper left coordinate and (w, h) means the rectangle size.

[6] An OpenCV function to detect objects of different sizes in the input image. The detected objects are returned as a list of rectangles.

(c) Slice the detected face region. Most of the pixels of this sub-image contains information about the person's skin color. Thus, you can analyze the histogram of the detected region to select the best threshold for your detector.

(d) Generate the histogram of detected face region (see Exercise 3.02 in Exercises #03). Try to use different color spaces (e.g. RGB, HSV, YCrCb) to check which of them better represent the skin color. In your report, discuss about your experiments. *Hint:* Check the scientific papers "A Survey on Pixel-Based Skin Color Detection Techniques" and "Skin Detection using HSV color space" to get an overview about skin color representation.

(e) Finish the skin color detection process, applying a binary mask in the input image. This item is similar that you have used in Exercise 3.04 (available in Exercises #03).

- **Output:** Record three videos with the results of skin color detector. You should name these files as `Ex106_E01.mp4`, `Ex106_E02.mp4` and `Ex106_E03.mp4`. *Hint:* You can use JPEG images and videos captured from a web camera.

(f) *(Optional)* Create a new method to detect some object according your own approach. For example, a method to select the best threshold to detect a round red ball. In this case, what is the best alternative to replace the face detection for this approach? *OBS:* Bachelor students can ONLY discuss the proposed method in their reports.

- **Output:** Record three videos with the results of automatic object detector. You should name these files as `Ex106_F01_Optional.mp4`, `Ex106_F02_Optional.mp4` and `Ex106_F03_Optional.mp4`. *Hint:* You have to use video captured from a web camera.