Assignment 1 - part 1

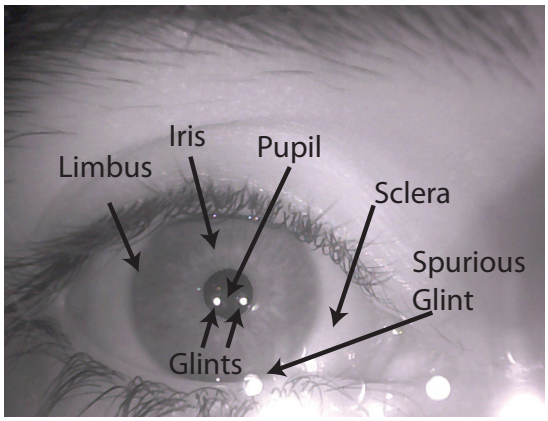

Figure 1: Various eye features

Purpose

This assignment spans several weeks. You will use some of the techniques taught during the course to make an eye and gaze tracker. You will/can be working with thresholds, image sequences, blurring, morphology, hough transformations. The assignment covers several exercises, but expect also to use some time outside class to finish it. Please follow the plan closely otherwise you will be busy the last week. Notice the second mandatory assignment is given before the deadline of this assignment.

Download and unpack *Assignment1.zip* from LearnIT. In this file you will find several image sequences containing eyes that you can use for testing your implementation. During the experiments you have to use a web camera to make your own datasets (e.g. sequences of your own eyes). You will be able to borrow the cameras from either Diako or Dan. Notice: use night vision (IR light) to get the best possible light conditions and clear features. For some cameras it is possible to cover the light sensors so the cameras use IR light – duct tape will be provided :D

The **report for this assignment** should be handed in groups of 2-3 students on the **26. March 2014** before the exercises. You will NOT have time the 12th and 26th March to work on the assignment since you will be given other exercises and assignment. Make sure the report contains:

- Your written report with **names** and **emails** on the front page
- The name of the file should be as follows: *Assignment1_GROUPINDEX*. (find your **group index** at LearnIT>assignment1> groups),
- A link to where the code can be downloaded
- The printed program.

Your report should NOT be a list of answers to the questions given in the exercise sheet. The questions should be considered as a guide that can help you to write a good report. The report should then either directly or indirectly answer the questions. Each report should be self contained and you have to describe what is the overall method (perhaps supported with figures showing individual steps) and what your assumptions are (e.g. glints are 200 pixels and have an intensity above 234). Remember to show that your implementation works (testing). This involves showing image sequences (multiple images), when the tracker works and when it does not work. A set of image sequences can be found in the *Sequences* folder. This means that you have to describe in words and pictures when the method works and when it is challenged. A set of image sequences will be given, but your report should contain sequences that you have made your selves. As with any other group work, it is important that you distribute the workload among the group members. So read through this document carefully and decide how you can work in the best possible way.

The **final report** (collection of all assignments) should be handed in to the examination office at the end of the semester. The final report should contain:

- Your written reports, perhaps arranged nicely :)
- A DVD with your report(s)
- Code
- Result videos and possibly test data.

Each mandatory assignment MUST be passed to be able to go to exam. Please see the guidelines for writing a report on LeanIT.

Remember that there is not only one solution to the assignment and we encourage you to be creative when solving it :)

About Eye trackers and Iris recognition

Iris recognition methods attempt to recognize the identity of the user based on the iris pattern. A gaze tracker, on the other hand, is a device, which is capable of determining where a user is looking. Both gaze trackers and iris recognition techniques (usually) use relatively similar image processing techniques for extracting useful information. Both initially detect features such as the pupil center, iris contour, white part of the eyeball (sclera) and perhaps the centers of IR reflections on the cornea (see figure 1). Based on the detected features the methods are able to determine gaze or the identity of the person (the latter steps are not a part of this assignment).

In this exercise you will detect specific eye features and if time allows you can experiment with more advanced methods. Notice that we will expect more work done by master students, but we are also flexible in letting the groups decide what this could be. While real-time performance can be obtained, then it is not speed that is the main objective for your implementation. It is better to have a solution that uses the techniques explained in the course in a sensible and novel manner.

Expected outcome for this week

Before next week it is expected that your implementation is at minimum capable of detecting the center of the pupil and the glints in a significant part of the provided sequences. There are several question this week that can help you in improving the implementation (e.g. to find more features or find the features more reliably). We encourage you to do as many as you have time for. Questions indicated with '*' can be skipped at first. Sections are also specified with numbers as to indicate their importance. If you have more time then work on eye corner detection where you will be using template matching (like in one of the exercises) to detect the eye corners.

Get code and Data

Download *"Assignment1_1 (Material).zip"* from learnIT (LearnIT). Extract the zip file and copy the contents into the source folder of your project. Open the folder in PyCharm.
The file *assignment1.py* contains the code skeleton for the assignment. You are naturally free to make changes to the file as you prefer. The file *SIGBTools.py* contains various auxiliary functions that may become helpful in solving the assignment.
Use the OpenCV reference manual [Ope11] or the online documentation for additional information and help about the OpenCV functions.

Questions and numbers

The text contains several questions that will guide you through the assignment. Sections marked with a ([1]) are required to do; higher numbers are optional but is intended to improve the overall report. Individual questions marked with '*' are not required (e.g. if time is short), but can help you to a better solution if time permits.

Strategy and Code skeleton

One common strategy for performing eye tracking is first to detect the pupil and glints and then the iris. The pupil and glints are in simple cases often determined through thresholds and connected component labeling. This should be your starting point. The iris can then be detected by using the pupil position as a starting point. The iris may, on the other hand, also be used to remove spurious glints and pupil candidates.

The overall skeleton of the assignment is given in the 'assignment1.py' file. The code contains methods to make sliders that you can manipulate (e.g. for thresholds, maximum and minimum areas etc. .) and some of the functions to be fully implemented. The subsequent steps are meant as a guideline for the assignment, but we urge you also to make your own experiments e.g. with intensity pre-processing of the images to enhance contrast (such as histogram equalization).

You will need to make your own image sequences for testing your methods. Each group can borrow cameras with IR light needed for the exercise.

1 Pupil, Glint and Eye Corner Detection

In this assignment (over several exercises) you will implement methods to automatically detect eye features. In particular:

- Pupil (dark region)
- Glints (the reflections of light sources on the cornea of the eye)
- Iris (a.k.a. Limbus)
- Eye Corners

The following steps should provide a guide to detect eye features. This assignment does not have one grand-solution, in fact, there is plenty of room to make individual implementations and testing out your ideas. There is a general guideline that you can follow. In the following it is assumed that the images are gray scale.

The output of this exercise: is a report and videos showing the location of eye features. Have in mind that we do not expect you to develop a method that identifies all eye features in every image sequence automatically, nor do we expect your method to be flawless. However, we do expect that the most of the features are found in most of the frames. This means that you can assume some static values for a specific image sequence (e.g. changing the threshold slider for each sequence). In your report you should describe what you did, why it works / why it fails. (Of course it is allowed to be ambitious, and try to develop generic and flawless methods - this is hard).

The report should contain images with the results of your method. Show several images of both when the method fails and when it gives good results. Use figures and the images to describe the methods and to analyze the methods.

Remember that the final report that is handed in to the examination office MUST contain video sequences of your results on a digital medium.

The pupil is usually the easiest to detect, so we start by finding it. The glints are usually not that difficult to detect either, however there may be too many good candidates.

Pupil Detection[1]

1. Run 'assignment1.py'. This should show three windows (one with the input image, one with the sliders and a binary image and a window that can be used to show temporary results). Move the *Stop/Start*slider to

the right to start running the sequence with the given settings. Try to characterize the appearance of the eye features (pupil, iris, glint and eye corners) in the sequence while it runs.

2. Rerun 'assignment1.py' Change the *pupilThr* slider so that the figure shows the pupil as clearly as possible while removing as much as the other structures as possible.

   The first step to find pixels that may belong the pupil and glints. You have to implement several functions that given a gray scale image detect these eye features. These methods will in this exercise imply using thresholds and connected components. Start by using thresholding and then progressively improve your method. Several suggestions are given at the end of this document.

3. Move the *Stop/Start*slider to the right to start running the sequence with the given settings. How well does the threshold find the pupil pixels? Move the slider back (left) to pause the sequence (or press space).

4. Change the filename of the input sequence (variable '*inputFile*')and run steps (2-3) with one or two of the other sequences located in the 'Sequence' folder. How well does the same size threshold apply to the other sequences?

   'Assignment1.py' has a function *GetPupil* which given an image and a threshold (and perhaps later also other parameters) should locate the positions of good pupil candidates and return them in a list. Ideally this list only has a single element, though, in practice the list may in some frames have more than one element. Your representation of the pupil should be the five parameters defining an ellipse $(x, y, r_1, r_2, angle)$.

5. You will need to use connected components and properties associated to each blob for this exercise. In a python prompt type

   ```
   >>>import SIGBTools
   >>>help(SIGBTools.RegionProps)
   ```

   and read the documentation and specifically look at the example within the documentation.

In the following you have to implement the function *GetPupil* so that it returns the pupil candidates. The method already calculates blobs based on a thresholded image. The threshold can be changed by moving the "pupilThr" slider.

6. Extend *GetPupil* so it iterates though each of the blobs and calculate region properties using *CalcContourProperties* (defined in *SIGBTools.RegionProps*).

7. *GetPupil* already draws a small circle in the image at a fixed position. Update *GetPupil* so that it draws the centers,C , of each all the detected blobs. The center is obtained through the 'Centroid' property in *CalcContourProperties*. The center,C, can for example be displayed in a color image as a small circle using *cv2.circle(tempResultImg,C, 2, (0,0,255),4)* This may be a bit slow, but don't worry it will be taken care of soon.

In the following steps you will filter the connected components (a.k.a blobs) based on regions properties. After the connected components have been filtered you will (perhaps) fit an ellipse around those candidates. The set of ellipses will be the result of this part of the assignment.

Region properties can be used to specify classification rules used to determine whether the blob corresponds to the pupil. A good start is use use the size of the contours to determine which blobs correspond to the pupil (a.k.a classification rule).

8. The pupil size varies, but only within certain limits. Define rules (i.e. 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. Area values between 500-6000 could be a good starting point. The lecture slides from last week has an example that could help you to perform the blob filtering that you could adapt.

9. Instead of drawing all the blobs, make *GetPupil* only draw the centers of the blobs that fulfill your classification rule. Don't worry if it is not perfect yet. You probably have to run your program a couple of times until you have found good candidates for areas that result in good pupil candidates.

10. Instead of rerunning the program you can use the sliders (minSize and max size) to set the threshold values for the classification rule. Update the signature of *GetPupil* function so that it takes the min/max area thresholds values as parameters.

11. Change the *update* function so that it calls *GetPupil* with the slider values for the min/max thresholds. The values can be obtained via the sliderVals (e.g. `sliderVals['minSize']`). See the function *getSliderVals* for further details.

12. The Assignment1.py has already created a window called "Temp". Make your program show the original original image and the centers of the blobs

13. Adapt your classification rule to include 'Extend' from *CalcContourProperties* (in addition to 'Area') to improve the results. Values less than one should give pretty good results.

Good pupil candidates can be represented as an ellipse. An ellipse is defined by 5 parameters (see figure 2) (x_0, y_0, a, b, θ), the center (x_0, y_0), the longest axis a, the shortest axis, b and the angle θ between the x-axis and the longest axis, a.

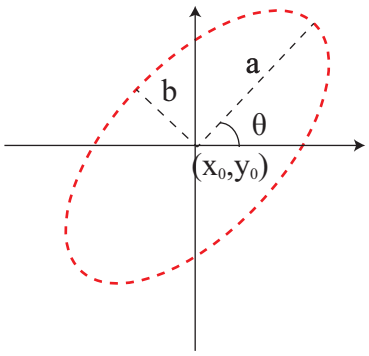

Figure 2: Parameters of an ellipse

14. Use *pupilEllipse = cv2.fitEllipse(cnt)* to estimate the ellipse that fits the contour points for those contours that are considered good according to the classification rule. You will hear more about contour fitting in the next lecture. It suffices for now just to use *cv2.fitEllipse(cnt)* on the connected component, *cnt*.

15. Make *GetPupil* return the list of ellipses corresponding to the good pupil candidates.

16. Uncomment the lines in the function *update* so that the pupil locations can be displayed in the original image. You may also want to show the circle center using *cv2.circle(img,C, 2, (0,0,255),4)*.

17. * You can save the result of your detection into a video by pressing 's' in the slider window. A message "Recording...." should be printed.

Morphology

Morphology can improve the detection results of the pupil (e.g. removing glints, filling out holes and avoid merging with iris pixels).

18. Which morphological operators would be best suited if you want to avoid the pupil pixels to merge with iris pixels?

19. Which morphological operators would be best suited if you want to remove the reflections in the pupil area?

20. Extend *GetPupil* function so that it uses morphology on the thresholded image to give better detection results for the pupil.

21. Experiment with the other sequences in the 'Sequence folder' by changing the filename string *inputFile* by changing the threshold values and other parameters settings as to improve the results. Which values (if any) can be used

22. Extend your code so that you can save the results in a new image sequence. This will be useful for your final report. Since it takes time every time you want to encode a new sequence, you can uncomment this code again after you have checked it works on a few sequence.

23. * Make additional sliders so that you can change your parameters interactively for pupil detection. You would need to update *getSliderVals()* and *setupWindowSliders()*

Glint Detection[1]

The next step is to locate the glints. Glints can be found using a similar strategy as with the pupil. The difficulty is, however, only to detect the true glints while reducing the number of *false positives*.

24. The function *GetGlints* has already been defined, but has not been implemented. Implement the function *GetGlints*, so that given a gray scale image and a set of parameters (now only size) it returns a list of glint candidates. Use thresholding, blob detection and blob filtering, just like for pupil detection, to get the reliable candidates. Start with just using 'Area' in the classification rule. Remember to uncomment the lines in the function *update* so that the glint locations can be displayed in the original image. You should expect to have several spurious glints detected.

25. 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. Run your program on the different sequences provided. For the report you should also make your own sequences using the web camera.

26. If you have not already done so, use morphology for glint detection.

27. * Make additional sliders so that you can change your parameters interactively for glint detection. You might by now have many sliders to manipulate.. but hey, this is just a prototype :)

28. Verify your method on different sequences in the 'Sequence' folder.

For the report In the report specify your assumptions on how you detect the blobs. Show the detected pupil and glint 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?

Using the relationship between eye features [1]

It can be difficult to set parameter values (e.g. thresholds) using only the blob (shape) properties of the individual features so that you always get perfect results. You will most likely have detected too many or too few glints and pupils in the sequences. At this stage it is better to have too many pupil and glint candidates as some of these can be removed based on their distances to other eye features. In the following steps you will use the spatial relations between pupil and glints to get more robust results. For this purpose the function *FilterPupilGlint* has been defined. In the following you will make changes (implement) the *FilterPupilGlint* function. Recall that the (Euclidean) distance between two point is:

$$\sqrt{dx^2 + dy^2}$$

, where dx and dy are the differences between the x and y coordinates of the two points.

29. Implement a function that calculates the distance between two pair of point.

30. Make your program load the *Eye1.avi* and run it (e.g. without glint and pupil detection). Observe how the pupil and glints move in the image sequences. For example, the two glints that are useful in this assignment, have roughly the same distance between them. This constraint can be used to define another classification rule that throws away those glints pairs that do not have the 'right' distance between them. In *FilterPupilGlint* implement such a classification rule and let the function return pupil and glints that fulfills these constraints.

31. Another rule/constraint that can be applied, is to use the distance between the pupil and glints, to remove those features that are too far away from each other. Define another classification rule that uses the distance between the pupil and glints to remove glints and pupil candidates.

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

33. In the report discuss whether and when such rules are useful and what the challenges are.

Eye Corners[2]

The eye corners are features defined over a small region that may be detected through template matching.

Eye corner detection is not mandatory for this week. In previous exercises you implemented template matching. This exercise is about using template matching for eye corner detection.

In you program ('Assignment1.py') you can select a template by pressing 'm' when the the sequence is paused. A new window appears on the screen where you can select an eye corner region (left and right clicks to select the region and return/space to accept).

34. For this exercise you can start without doing pupil or glint detection. Start by commenting the lines in the *update* function that call the pupil and glint detection functions.

35. Change the program so that it loads the image sequence *Eye8.avi*.

36. Key presses are handled in the *run()* function. Extend the *run()* function so that you are able to select two regions by using two calls to the *roiSelect.SelectArea* function.

37. Store the templates corresponding to the two regions in the global variables *leftTemplate* and *rightTemplate*.

38. The function *GetEyeCorners* takes two eye templates as input. These should be used for template matching. Use template matching to detect the left eye corner in the image sequence using template matching. Hint: You may be inspired by the solution in the exercise 4.

39. Do the same for the right eye. Are there any difference in how the methods work on the left and right eye corners?

40. In the report you should discuss the influence of different regions sizes and which template method performs the best (time and location)

Ideas for further improvements

41. [3] 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. Extend the *GetEyeCorners* function so that it is able use the pupil position to constrain the the search for the optimal location of the eye corner templates.

42. [4] How could only one template be used?

43. [4] Use an image pyramid to improve search.

44. Experiment with several sequences to be able to document the properties of using template matching for eye corner detection.

2 Ideas and questions for improving the tracker[3]

There are many ways to improve the results (time and accuracy) for both glint, pupil and eye corner detection. Chose as many of these as you have time for and not necessarily in this order. Please don't hesitate to ask if you want some guidelines for how to proceed with these.

- What effect does histogram equalization have on the eye images? Can it be used to improve or simplify your method? (difficulty: Simple)
- Experiment with using (binary) morphology (open, close, dilate and erosion) on thresholded pupil and glint images. Can the detection of pupil and glints. be improved with using morphological operators after thresholding steps? (difficulty: Simple)
- Use Laplacian detect the glint (difficulty: Simple)
- Use Haar features filter the results. (difficulty: Medium, may take some time, though)
- Using image pyramids for eye corner detection (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 features in a sub region of the image. racking: use the previously found locations to filter those candidates that are too far away. (difficulty: Simple)
- You can use the eye corners and iris to detect the sclera. Implement a method that does this. Could we have used the sclera to detect the other features? (Diffiuculy. Medium)

References

[Ope11] *The OpenCV Reference Manual*. 2011.