2.1.2019

# Analysis and processing of biometrics images

Report 2

Dominik Gonciarz

## Introduction

Algorithm I've implemented bases on two separate methods for finding inner and outer circles around iris. Both identify areas utilizing the characteristic colors of an eye. I've assumed that the pupil is the biggest black region of circular shape close to the center of a photo. each of those assumptions may sound a little bit naïve and may produce irrelevant results for specific images (dark, acentric), combining them hopefully leads to correct result. Proper detection of pupil is significant to obtain accurate result since, second part detecting sclera, assumes correctness of center point.

## Running instructions

The algorithm is implemented in python as command line program. To run it requires python installation with all packages listed in requirements.txt

To run, execute:

```
python main.py --img <path to eye image>
```

## Steps:

1. Initial
    a. Load image
    b. Blur it applying gaussian filter, twice
    c. Enlarge color scale using histogram expansion
2. Detect pupil
    a. Apply minor contrast
    b. Convert image to grayscale
    c. Reduce colors in image to two, all values which all not almost black are converted to white, rest becomes black:
    ```
    c – value of color
    c > 10 ? c = 255 : c = 0
    ```
    d. Dilate image, with kernel of size 9
    e. Erode image, with kernel of size 9
    f. Find regions
    g. Guess which region pupil is
3. Detect sclera
    a. Apply noticeable contrast
    b. Convert image to grayscale
    c. Reduce number of colors in the image to at most 4
    d. Threshold image splitting color palette in half
    e. Erode
    f. Dilate
    g. Estimate radios of outer circle

## Dilate image:

For each pixel in an image take array if its neighbors. If any is white, change color of the pixel to white. This operation is applied to close inner spots, close regions.
In this case kernel size means the size of considered array of neighbors.

## Erode image:

For each pixel in an image take array if its neighbors, number can be defined. If any is white, change color of the pixel to white. This operation is applied to remove small spots of color, or in my case to revert unwanted change in radius done by dilatation.
In this case kernel size means the size of considered array of neighbors.

## Find regions:

This method is well explained in the pdf, understanding the description of implementation is easier after taking a glance on this: https://web.cs.wpi.edu/~emmanuel/courses/cs545/S14/slides/lecture08.pdf

In this method two arrays of the same size w/h are used.

```
img - image array, source

reg – use for calculations, it stores detected regions.

regions_count = 0


for all pixels p in img

      If p == black //region

            t = reg[p.x, p.y - 1]

            l = reg[p.x – 1, p.y]

            Assign to reg[c, r] the smallest but greater than 1 of ‘t’,
            ‘l’. If both are smaller or equal to 1 then assign
            regions_count++.
```

Whenever ‘u’ was not equal to ‘l’ and algorithm chose smaller, there is a conflict. One connected region has two subregions marked with different labels. In all such cases algorithm renames region with higher mark to match lower.
In the last step labels are scaled for better visualization.

## Outline regions:

The function takes as an input image array with regions marked as different values of gray.

It returns list of properties of regions detected in the image and image array with regions filled in solid color.

Algorithm loops through all pixels in the image and whenever it encounters region with label which has not been analyzed yet it calls the function to follow region border. From returned list it extracts, boundary rectangle, size, perimeter.

## Follow region border:

Takes as input coordinates of starting point, array of regions and array of filled regions.

It returns list of pixels outlining region.

It loops through rows of the image starting from provided point and going down then, up. In each row it finds first and last index of pixel having the same label as starting point. It fills everything between with some constant color and adds bordering to the list.

## Guess pupil

Function tries to find which of outlined regions can be pupil

It sorts regions according to length of border.

Size of region is compared to the total size of an image. If the region is smaller than one percent, it is rejected. Then function assumes shape is close to circle, radios is calculated using three different methods: from known area, diameter, and perimeter. If a shape is circular all should be similar. Also, distance from the center of the image is taken as a factor.

## Quantize colors

Quantization I've implemented is like uniform. The only difference is that usually to generate the palette of colors from an image only unique colors are considered. My implementation creates long list of all colors (possibly not unique), sorts according to value of color and splits into number of intervals of equal size. Then calculates average for each interval and repaints image using generated palette.

## Estimate outer radios

In black and white image find pairs of pixels where one is white, second is black. Take first and calculate distance to given center of an eye, append this value to the list. When there is more than 160 percent of height of an image adding. Sort list. Remove 10 percent of the smallest and 10 percent of the greatest. Calculate average and take it as radios.

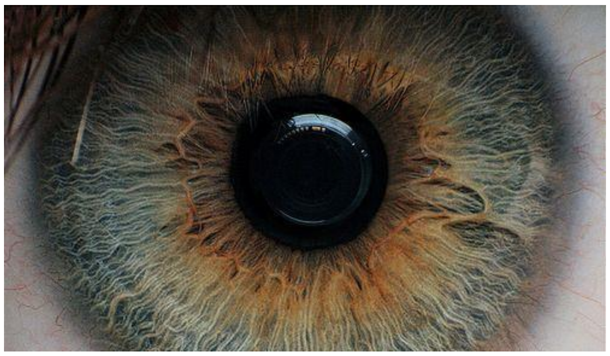
## Accuracy and efficiency

The algorithm properly detects iris in the provided image, however whole process takes enormously long time. Performance is not really an issue for only one image but before anyone would like to use it for larger dataset significant optimization needs to be made. Program loops through whole image way too many times, creates many copies of image in different states and separately calculates properties which should be attached to some of previous loop.

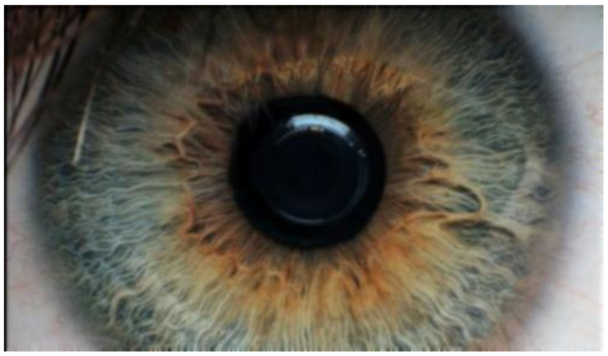
Such fragmentation of code is useful for debugging purposes, when developer needs to be able to see changes after each part of instructions.

## Visualization of steps:

### Input

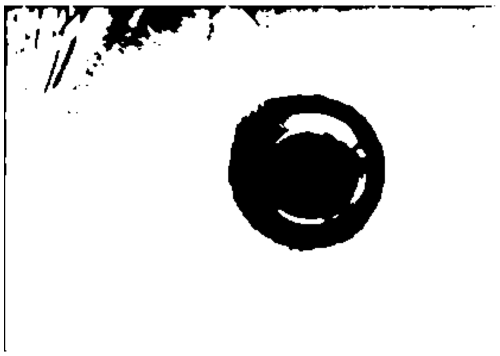


### Blur image, expand histogram

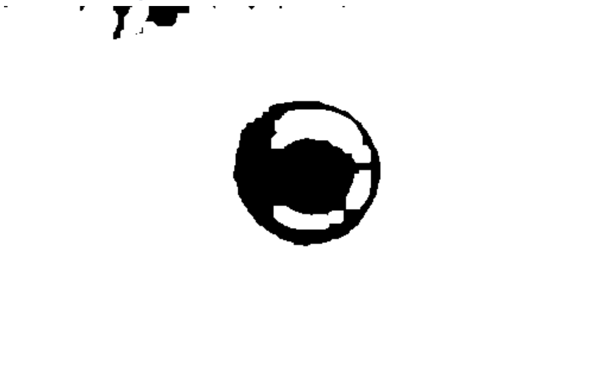


### Detect pupil

#### Apply minor contrast, Convert image to grayscale, threshold

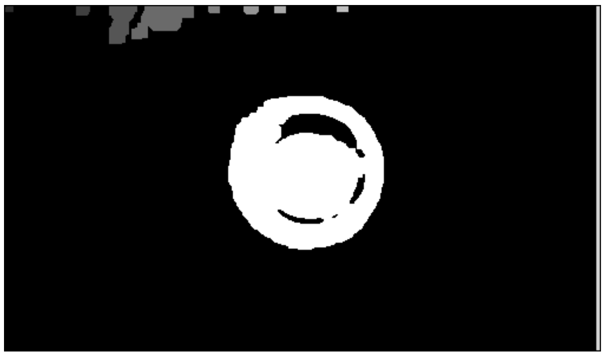


Dilate image, with kernel of size 9

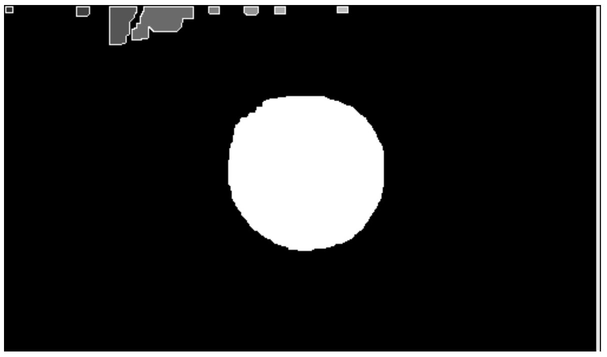


Erode image, with kernel of size 9

Find regions, color them

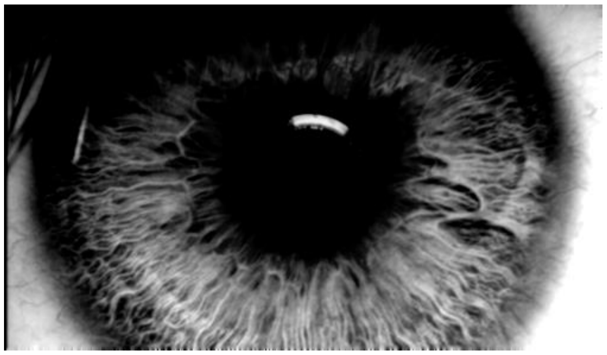


Fill regions, outline

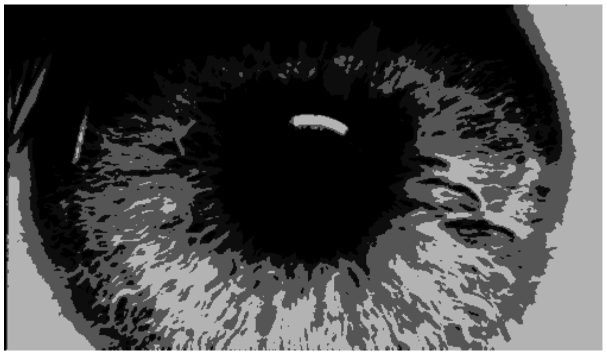


## Detect sclera

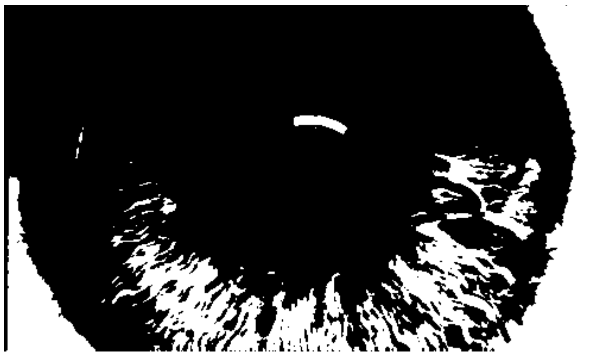
### Contrast, grayscale



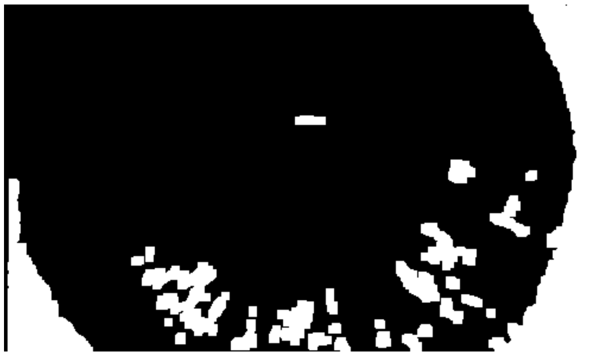
### Quantize



### Threshold



### Dilate, erode



## Result