# Lab 7 & 8 - Retina Vessels

## Classical Retinal Vessels segmentation

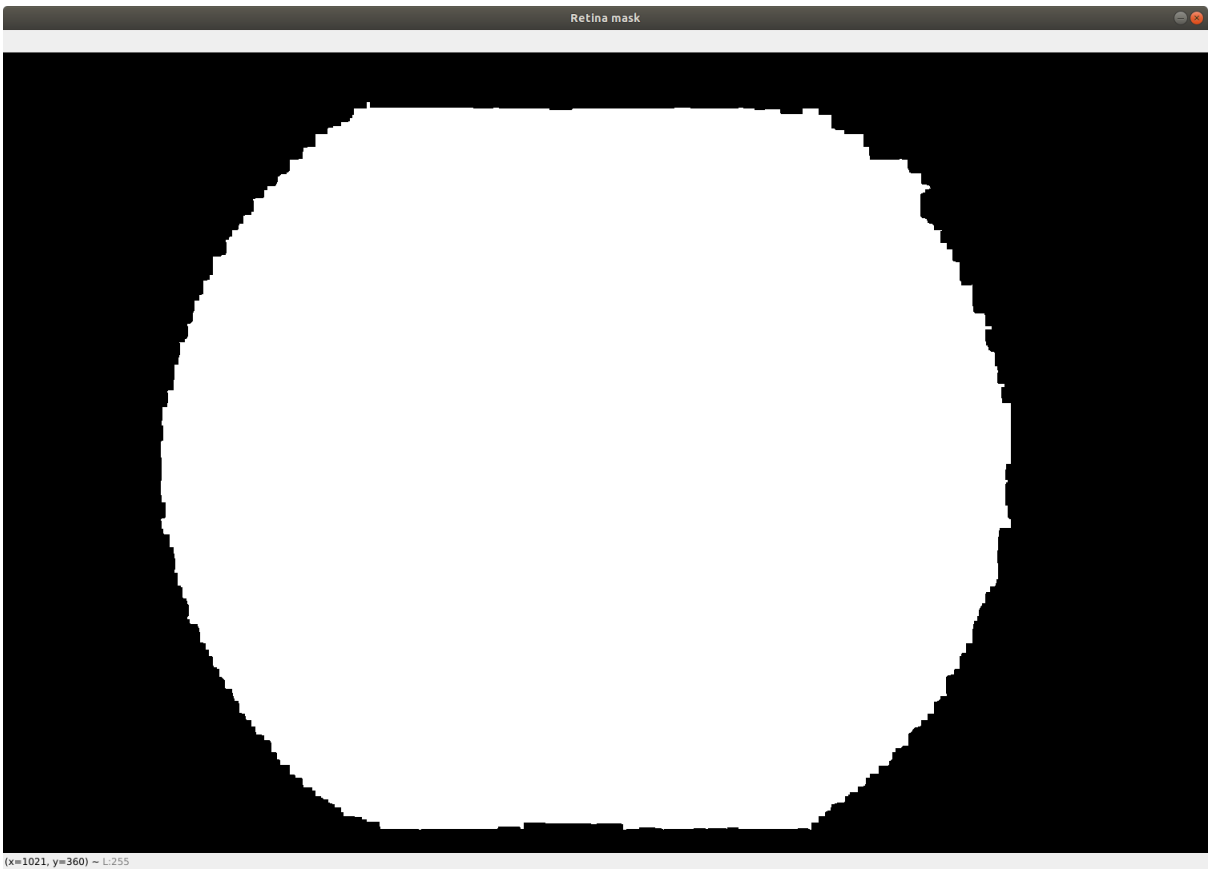
We will use method inspired by:
Zhi-Wen Xu, Xiao-Xin Guo, Xiao-Ying Hu and Xu Cheng, "The blood vessel recognition of ocular fundus," *2005 International Conference on Machine Learning and Cybernetics*, Guangzhou, China, 2005, pp. 4493-4498 Vol. 7, doi: 10.1109/ICMLC.2005.1527730.

Fully classical approach in this case is quite tricky and demanding. Therefore we will implement only the segmentation part after which we will obtain the skeleton of retinal vessels.

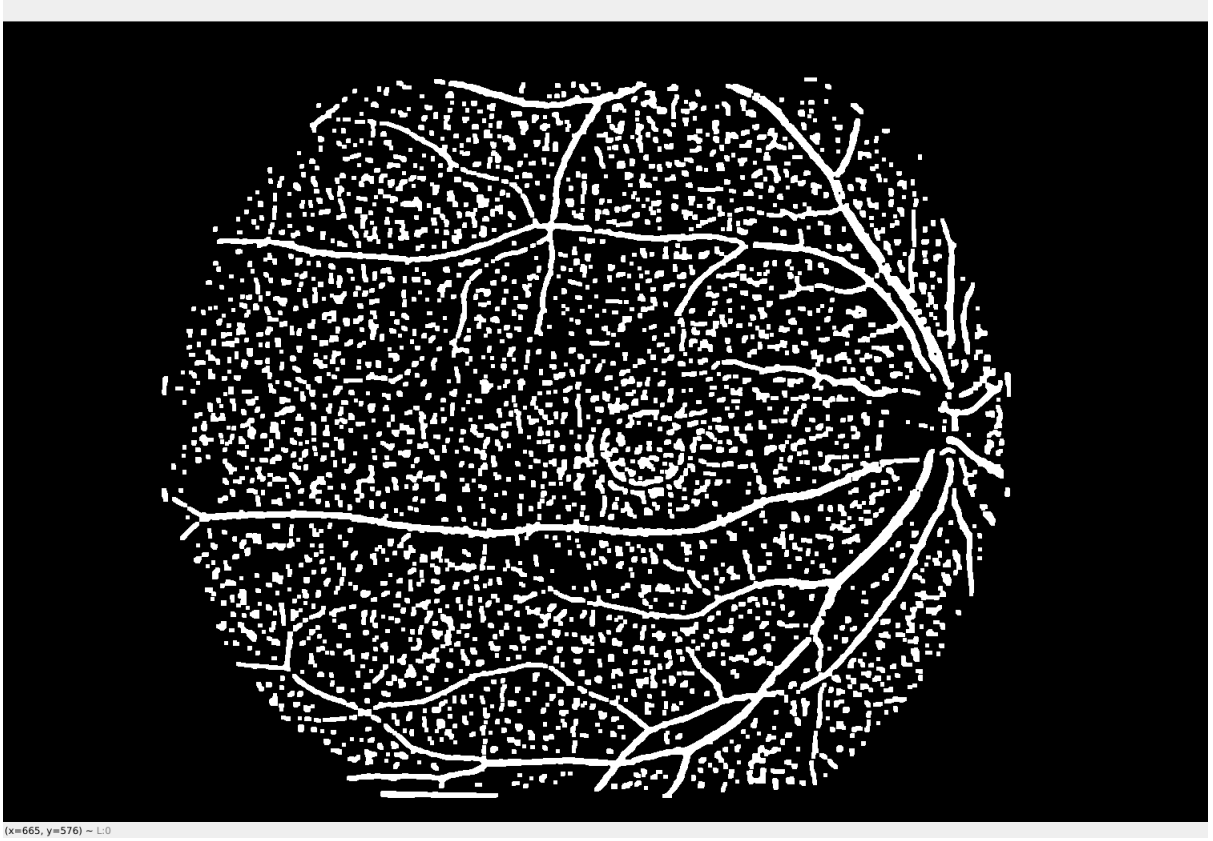
Download RIDB dataset:
https://drive.google.com/drive/folders/1FtIX04PLdeDe7I_n8zG2xhzVngwnnlLU?usp=sharing
As you can see, it consists of retinal images gathered from 5 different eyes (20 images for each). Remember that our task is not to match individuals on the basis of these photos, but only to segment blood vessels.

To do so, create a new Python script from scratch. For each image in RIDB dataset implement the following steps:

- convert image to the grayscale
- prepare mask to remove the black border from the image with the use of floodfil algorithm (cv2.floodFill) from the point (0, 0):
    - remember to work on the gray image copy:
      `img_floodfill = img_gray.copy()`
    - to obtain proper mask you can use the following code:
      ```
      h, w = img_floodfill.shape[:2]
      floodfill_mask = np.zeros((h+2, w+2), np.uint8)
      ```
    - dilate (cv2.dilate) the floodfill result to cover pixels near the ROI (retina image) - use kernel of size 11 or 13:
      `dil_kernel = np.ones((13, 13),np.uint8)`
    - finally, do the bitwiste_not and binarization with small threshold (e.g. 1) to obtain the proper mask - it should look similar to the one below:




- enhance contrast of the grayscale image using CLAHE algorithm (`clipLimit = 2.0, tileGridSize = (8,8)`)
- invert enhanced grayscale image (simply: 255 - img) and once again use the CLAHE algorithm
- blur image with the gaussian kernel - cv2.GaussianBlur (size 7x7, sigma calculated from size)
- do the adaptive thresholding with gaussian-weighted sum of the neighbourhood values (cv2.ADAPTIVE_THRESH_GAUSSIAN_C, C = 0)
- combine threshold output with previously prepared mask (cv2.bitwise_and)
- blur image with the median filter - cv2.medianBlur (size 5x5) and invert values (cv2.bitwise_not)
- fill the holes with morphological close operation - you can use kernel from previous dilation
- again invert the image - the output should look similar to the one below:




- get objects stats and remove background object with [Connected Component Analysis]:

```
nb_components, output, stats, centroids = cv2.connectedComponentsWithStats(img_closed_inv, connectivity=8)
sizes = stats[1:, -1]
nb_components = nb_components - 1
```

- remove smaller objects (area < 500)
- finally, get the skeleton of the preserved objects - you can use the following code (`img_obj_filtered` is the input image and `skel` is the output skeleton):

```
skel_element = cv2.getStructuringElement(cv2.MORPH_CROSS, (3,3))
skel = np.zeros(img_obj_filtered.shape, np.uint8)
while True:
    opened = cv2.morphologyEx(img_obj_filtered, cv2.MORPH_OPEN, skel_element)
    temp = cv2.subtract(img_obj_filtered, opened)
    eroded = cv2.erode(img_obj_filtered, skel_element)
    skel = cv2.bitwise_or(skel, temp)
    img_obj_filtered = eroded.copy()
    if cv2.countNonZero(img_obj_filtered)==0:
        break
```