Medical Image Processing – Exercise 5

Submitted by: Alina Ryabtsev (cse username: alina.ryabtsev), 208696054

General Notes:

Design:

Part A of this exercise was implemented in *RetinaRegistration.py* file and its functions are in *RetinaRegistration* module, while part B of this exercise was implemented in *RetinaDifferencesDetecor.py* file. Also "*ransac"* function *utils.py from previous exercise* was used in Part A for automatic point registration. *main.py* file runs the program.

Libraries used:

- scikit-image – for processing images (morphology, measure, filters, segmentation, transform)
- matplotlib – for plotting graphs and showing images
- numpy – for matrices operations

Part One – Retina Registration:

Module: RetinaRegistration.py

Registration by feature points:

execute_registration (self, registration_type) – This function performs transformation according to given registration type.
Input: ▪ registration_type – if 1 then performs registration by feature points, 2 by blood vessels segmentation.
Output: None

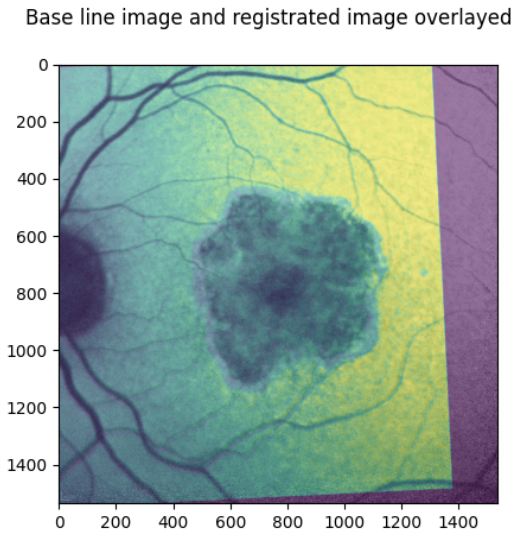
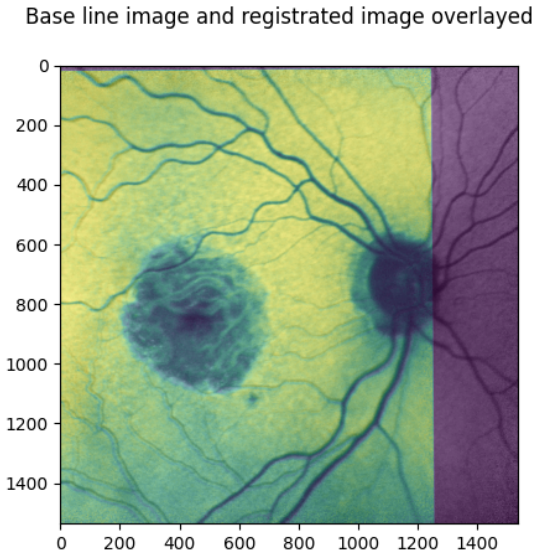
find_retina_features (self, is_dubug) – finds strong features (key points and descriptors) in both follow up and baseline images with the ORD detector algorithm.
Input: ▪ is_debug – shows graphs for debugging purposes.
Output: None

brute_force_matcher (is_dubug) – This function performs brute-force key points matching on both baseline and follow up images. It takes 15% of the matches as the best matches found with the brute-force detector.
Input: ▪ is_debug – shows graphs for debugging purposes.
Output: None

calc_point_based_reg (self, bl_points, fu_points) – This function returns transformation matrix of the translation and rotation of the follow up image. It uses SVD in order to find the optimal transformation.
Input: ▪ bl_points – baseline points ▪ fu_points – follow up points
Output: a 3x3 rigidReg rigid 2D transformation matrix
Helper functions: ▪ _compute_weighted_entroids(self, bl_points, fu_points) ▪ _compute_centered_vectors(self, bl_points, fu_points) ▪ _comute_covariance_matrix(self, bl_points, fu_points) ▪ _optimal_rigid_transformation(self, bl_points, fu_points)

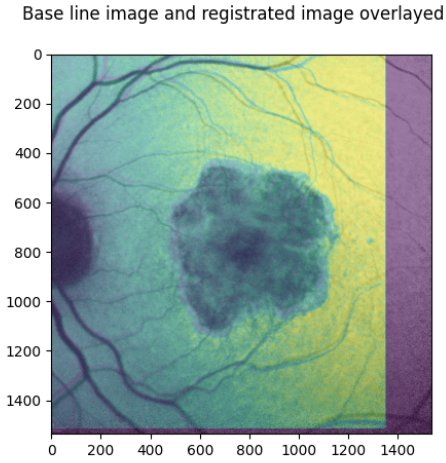
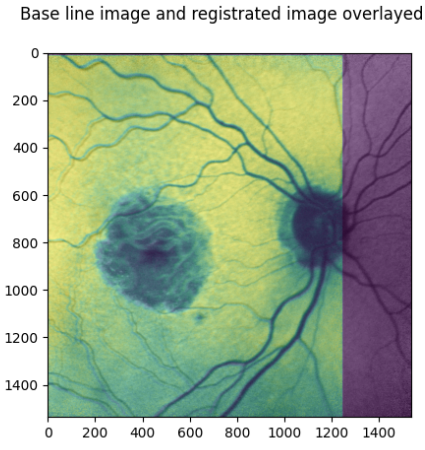
calc_dist (bl_points, fu_points, reg) – computes the distance of each transformed point from its matching point in pixel units.
Input: ▪ x – baseline points ▪ y – follow up points ▪ reg – rigid registration

Output: a vector of length N which describes the RSME

compute_registrated_image_by_features (self) – computes the transformed image the follow up image after registration and shows it.
Input: None
Output: the baseline and follow up images after registration.
Image 1: *FU01.tif image overlayed on BL01.tif image after applying registration on it:* 
Image 2: *FU03.bmp image overlayed on BL03.bmp image after applying registration on it:* 

Registration by blood vessels segmentation:

segment_blood_vessels (self) – This function performs blood vessels segmentation of the provided baseline and follow up images by running some thresholding methods.
Input: None
Output: None

compute_registrated_image_by_vessels_segmentation (self) – computes the transformed image and the follow up image after registration and shows it.
Input: None
Output: the baseline and follow up images after registration.
Image 1: *FU01.tif image overlayed on BL01.tif image after applying registration on it:* 
Image 2: *FU03.bmp image overlayed on BL03.bmp image after applying registration on it:* 

Part Two – Detecting the differences of two images:

Module: RetinaDifferencesDetector.py

Differences Detection:

__init__(self, registrated_bl_image, registrated_fu_image) – Initializes instance of the differences detector. The images provided are normalized if necessary.

Input:

- registrated_bl_image – registration of the baseline image of the retina.
- registrated_fu_image – registration of the follow up image of the retina.

detect_differences (self) *– This function performs detection of the changes of the two images.* *Algorithm flow:* *First, the images are subtracted, so the differences between them are obtained in one image. All values greater than 0 are lowered to 0 and the absolute value is taken. Next, thresholding action is performed. Skimage library offers various thresholding methods, which lead to different segmentation results. For example:*

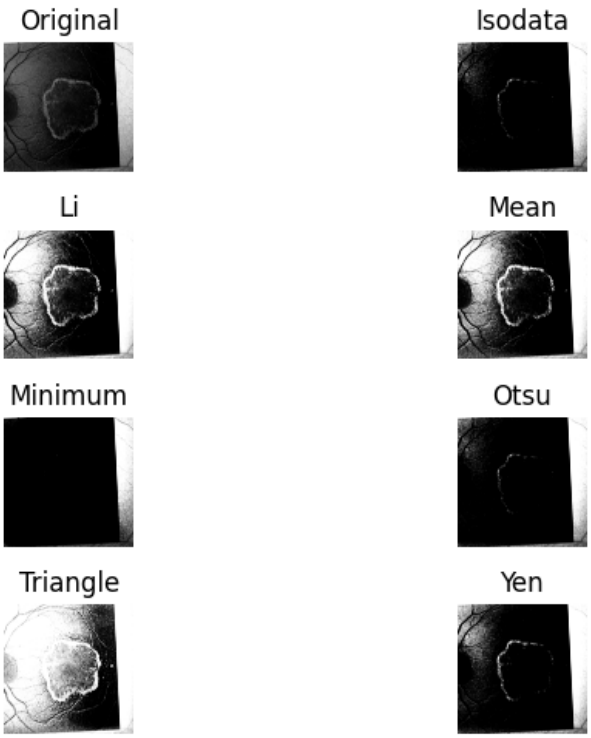


Figure 1: Skimage different thresholding methods performed on BL01.tif and FU01.tif images overlayed

Note that "Li" / "Mean" thresholding methods perform a good segmentation of the leisure growing, although a lot of "noise" as darker background was caught either. Meanwhile, the "Yen" thresholding method catches less noise but does not detect all the changes. So, taking an average threshold of them generates quite good thresholding for the image.

After getting a binary image, some morphological operations were performed. First, dilation was performed, before running skimage.segmentation.clear_border() operation. The dilation led to better results here, and a clearer segmentation was received at that step, but still a little noisy. Then, skimage.morphology.remove_small_objects() was performed, which helped a little with the noise. Afterward, erosion was performed, some median smoothing, again removing small objects and another median smoothing. Finally, closing operation with disk size of 20 was performed and precise segmentation with a little noise was received.

Input:

- None

Output:

- None

Image 1: *BL01.tif and BL01.tif images after running differences detection algorithm*

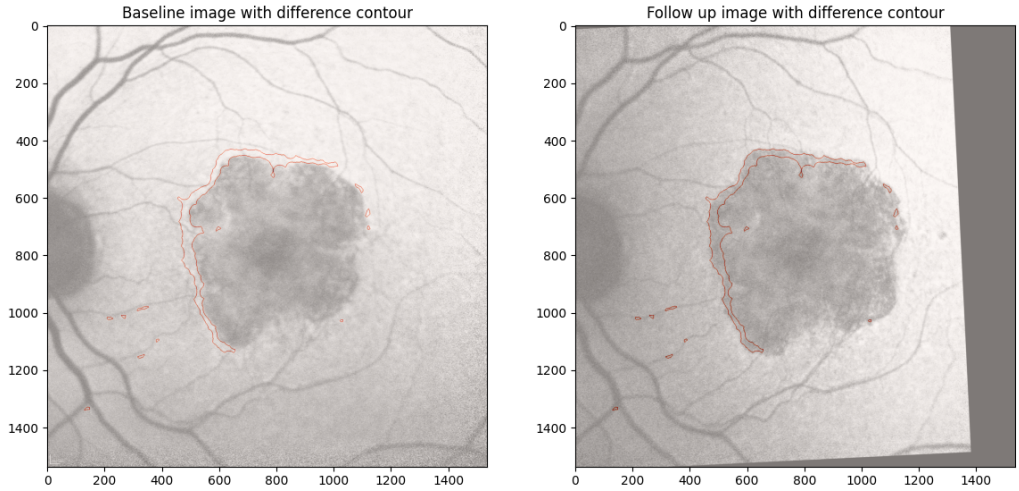
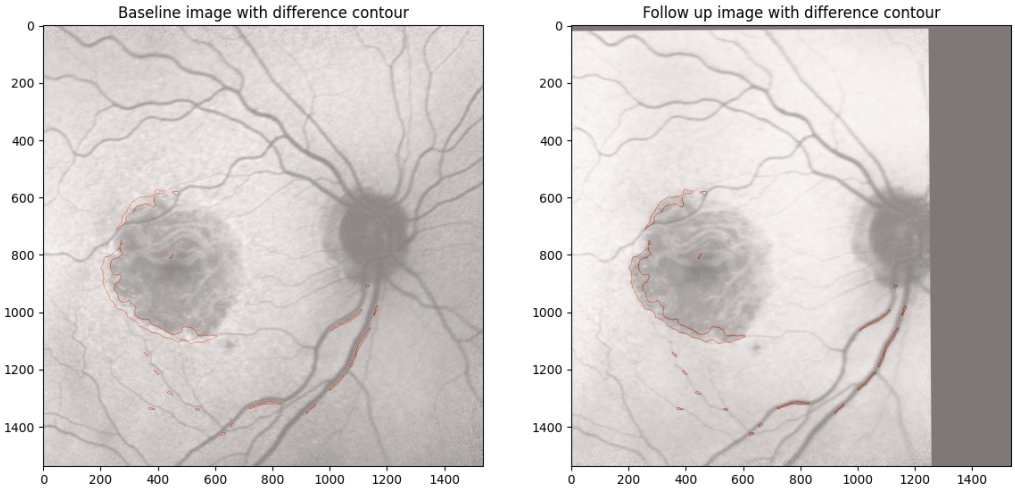


Image 2: *BL03.bmp and BL03.bmp images after running differences detection algorithm*


Baseline image with difference contour
Follow up image with difference contour
0
200
400
600
800
1000
1200
1400