Universitat de Girona

MAIA

UNIVERSITAT DE GIRONA, MEDICAL IMAGE REGISTRATION AND APPLICATIONS

Report: Rigid Image Registration | 2021-2022

OJEDA, Manuel; VASILE, Alexandru

November 26, 2021

1 INTRODUCTION

The aim of this lab is to develop competences regarding teamwork and problem solving. By developing the proposed activity you will also become familiar with SIFT and planar transformations for rigid image registration: how to extract invariant features, how to describe them, how to match them and how to use them to compute a homography. Most importantly, this should give you some feeling about the strengths and weaknesses of local feature-based approaches.

2 TEST LOWE'S SIFT VS THIRD PARTY IMPLEMENTATION

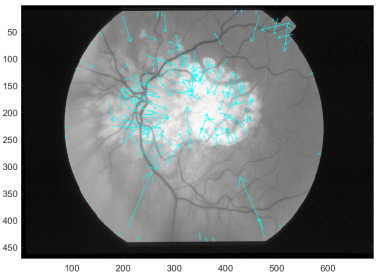
In this part of the report, not only the behavior of Lowe's implementation of SIFT is going to be studied, but also compared with a third party library. To display side-by-side the results obtained by both implementations and to explain why they might have differences and, if so, how the differences could be tackle in order to achieve a similar result by fine tuning of the parameters or to explain the reasons of these differences.

2.1 Lowe's SIFT algorithm

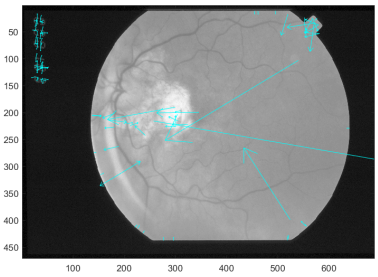
The first part consists on running Lowe's SIFT algorithm using the files downloaded from the website on MatLab. The algorithm is implemented on the images from the Dataset00. The purpose of this step is to understand how the algorithm works and to visualize results obtained. The Lowe's algorithm for Matlab is made up of 3 important files, each one realizes different functions which will be explained. The reasoning of these files were realized with the original values, without any modification, with the purpose of understanding the algorithm as it was established by Lowe; the modifications of certain values are realized afterwards, in the comparison section between Lowe's algorithm and the third party implementation.

2.1.1 *Sift function*

This file reads an image and returns its SIFT key points. Where the only input is an image, for this case, the user must pay attention to the kind of image set as input. Because the function considers every image as RGB image and converts it into a gray scale, so Matlab is going to send a warning if the image is already a gray scale. To solve this problem, either the input is an RGB image or the user must comment the line of the sift function where converts the image into a gray scale image.
The function returns 3 variables: *image, descriptors, locations*. The first one is the image as an array in double format. The descriptors is a matrix of *K-by-128* where *K* represents each one of the normalized vector of the key points found. The descriptors have a length of 128 due to the experiments ran during Lowe's experiments explained in his paper which he found it was the most efficient approach. Each one of the windows of 16x16 pixels, where the region is divided into 4x4 subwindows and the gradient orientation has 8 entries per each one of the subwindows. The final variable returned, locations, is a matrix of *K-by-4* where each one of the columns represent: *row, column, scale, orientation* for each one of the key points, *K*.



(a) Key points on the 1st image of the retina.

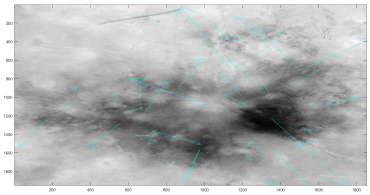


(b) Key points on the 2nd image of the retina.

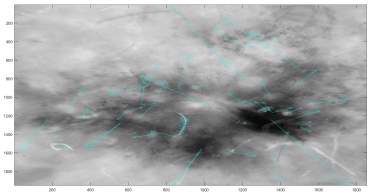
Figure 2.1: Key points shown for both images of the retina.

The figure 2.1 shows both images of the retina obtained from the Dataset00, after the sift function is run on them. For the figure 1(a), the amount of key points found is 240 while for the figure 1(b) is 112. One more thing to consider is the location of these key point, while for the first image the location of the key points are found within the region of interest, retina. For the second image, there are some key points outside of the region of interest, because the second image contains numbers on the upper left corner which the algorithm identifies as key points when in reality the key points of interested should be only located in the retina. This problem can be tackle once the match between those to images is realized, since there will not be any similar key point related to the numbers, those points will not be consider. Each one of the arrows represent every key point, the length and direction of the arrows represent the magnitude and orientation considered by the algorithm.

The figure 2.2 shows both images of the skin obtained from the Dataset00, after the sift function is run on them. For the figure 2(a), the amount of key points found is 73 while for the figure 2(b) is 100. Unlike the previous figure, all the key points found for those two images are within the region of interest.



(a) Key points on the 1st image of the skin.



(b) Key points on the 2nd image of the skin.

Figure 2.2: Key points shown for both images of the skin.

2.1.2 Show key points function

The file displays an image with SIFT key points overlayed. The inputs required are the image and the location matrix, both variables are obtained from running the previous function. This function was used to display both figures, 2.1 and 2.2. Before drawing the line for each one of the key points, the function first translates, rotates, and scales the line according to the key point parameters.

2.1.3 Match function

This file might be considered as the most completed, because with this file you can detect, match and displays lines connecting the matched key points. The only necessary inputs are the two images and the figure will be displayed. The function runs the files previous mentioned to obtain the final result, and A match is accepted only if its distance is less than *distRatio* times the distance to the second closest match, the value in the original algorithm is set to 0.6. The optional return value is the number of matches displayed.

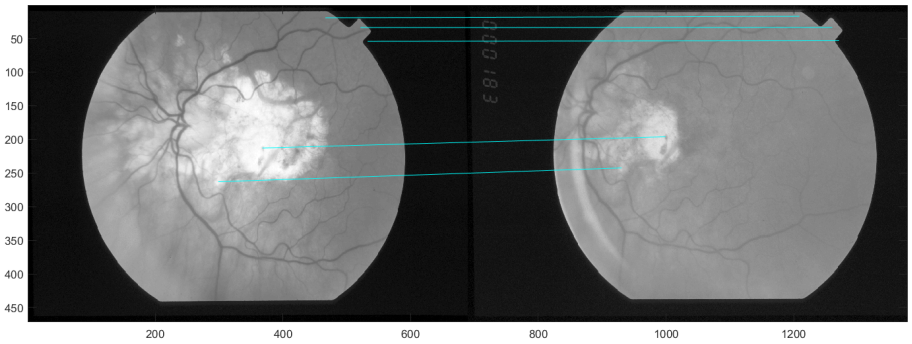


Figure 2.3: Matches between the two retina images.

The figure 2.3 shows the matches between the two retina images from the Dataset00, the number of matches is 6. Once the image is analyzed, one of the matched points is outside the region of interest, retina, but due to the descriptor itself and distance between the two points is considered a match.

The figure 2.4 shows the matches between the two skin images from the Dataset00, the number of matches is 19. The images themselves present some differences, because they were acquired with different type of cameras, Canon DSLR and Nikon DSLR, but they were taken using the

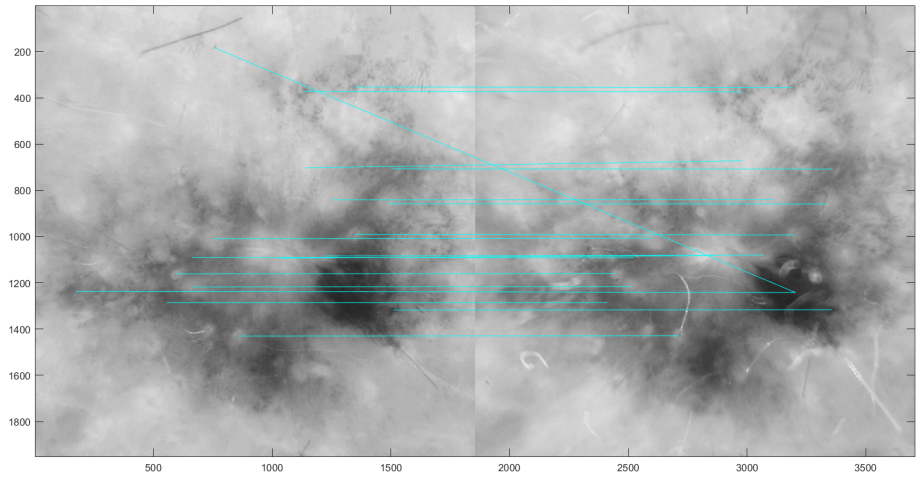


Figure 2.4: Matches between the two skin images.

same Dermlite dermoscope. All the matches, but one, can be correctly identified with a visual evaluation. The only match which can be considered as incorrect is the line with inclination, which does not follow the same parallelism as the others, and with a visual evaluation is evident it does not match with the point in the other image.

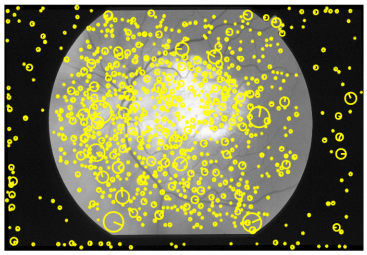
2.2 Third party implementation, VLFeat

The algorithm is implemented on the images from the Dataset00. The purpose of this step is to understand how the algorithm works and to visualize results obtained. In order to get the sift key points, match and visualize them, as in the Lowe's algorithms part, three functions are needed which will be explained. The reasoning of these files were realized with the original values, without any modification, with the purpose of understanding the algorithm as it was established by the third party; the modifications of certain values are realized afterwards, in the comparison section between Lowe's algorithm and the third party implementation.

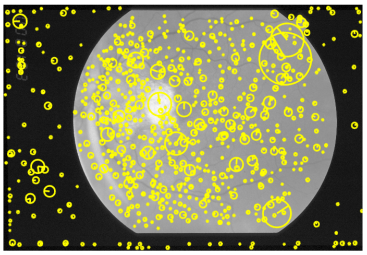
2.2.1 VL sift function

The only input for this function is the image, which has to be gray scale in single precision. Unlike the function from Lowe's algorithm, this one allows to decide how many outputs to have, either one or two. The mandatory one is *F*, it represents a feature frame containing the information about the location of the key points, its dimension is matrix of *4-by-K* where each one of the rows represent: *[X;Y;S;TH]* for each one of the key points, *K*, X and Y is the center of the frame, S is the scale and TH is the orientation (radians). The second optional output is *D*, it is a matrix of *128-by-K* where *K* represents each one of the descriptor of the key points found.

The figure 2.5 shows the same retina images used on the Lowe's algorithm, but now using the third party implementation. The first thing observed is the amount of key points detected. For the figure 5(a) the amount of key points is 1000 and for the figure 5(b) the amount is 842. The second thing to observe is the way is represented the key points, with circles and a line representing the orientation. Even though there is a big amount of key points within the region of



(a) Key points on the 1st image of the retina.

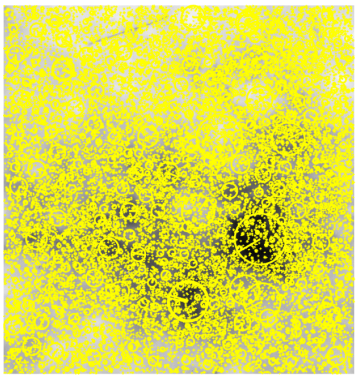


(b) Key points on the 2nd image of the retina.

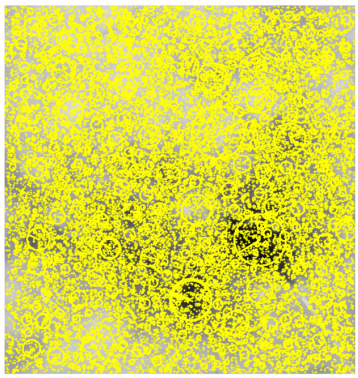
Figure 2.5: Key points shown for both images of the retina using algorithm of VLFeat.

interest, in the retina, there is a big amount of key points in the dark area.

The figure 2.6 shows both images of the skin obtained from the Dataset00, after the sift function is run on them. For the figure 6(a), the amount of key points found is 17961 while for the figure 6(b) is 19532. Unlike all the previous figures, with these two figures, the skin itself is almost impossible to see, because of all the key points identified by the algorithm.



(a) Key points on the 1st image of the skin.



(b) Key points on the 2nd image of the skin.

Figure 2.6: Key points shown for both images of the skin using algorithm of VLFeat.

2.2.2 Plot frame function

The function plots a geometric frame. The input requires the frame variable obtained from running the previous function. The frame can be either a 2D point, a circle, an oriented circle, an ellipse, or an oriented ellipse. Each one of those options need different amount of values from the FRAME, from two components up until six components. This function is used to visualize all the key points in figure 2.5 and 2.6.

2.2.3 Match function

This function is in charge of matching the descriptors, where the inputs are two descriptors matrix obtained from the sift function. It has an optional input which is a *threshold*, where a

descriptor from the first input is matched to a descriptor of the second input only if the distance multiplied by the threshold is not greater than the distance of the first descriptor to all other descriptors. The default value is 1.5.
The function has either one or two outputs, one is the matches obtained from the two sets of SIFT descriptors. The second output, optional, returns the squared Euclidean distance between the matches. The function uses the algorithm suggested by D. Lowe to reject matches that are too ambiguous.

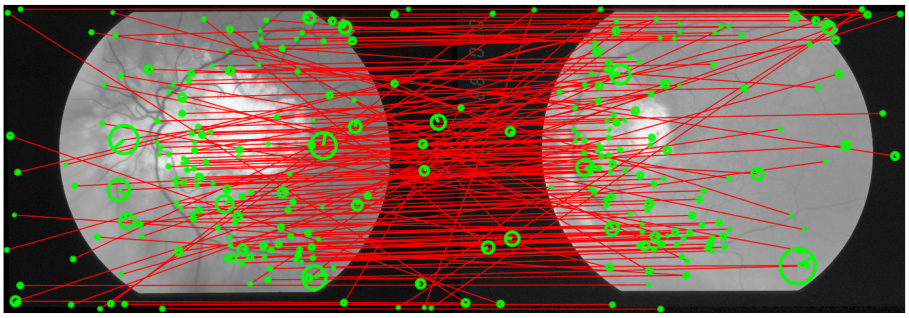


Figure 2.7: Matches between the two retina images using algorithm of VLFeat.

The figure 2.7 shows the matches between the two retina images from the Dataset00, the number of matches is 157. All those matches were obtained with the default values, the inputs used were obtained from the sift function. Where the initial amount of key points were between 1000 and 800, the final amount of matches went down to only around the 8% of the total amount of key points between both images. When in reality, the inliers are less than the 8% due to the outliers found in the matches by doing a visual examination. Some of the lines drawn have certain inclination or even seem to be almost perpendicular indicating they do not match their corresponding descriptor in the other image.

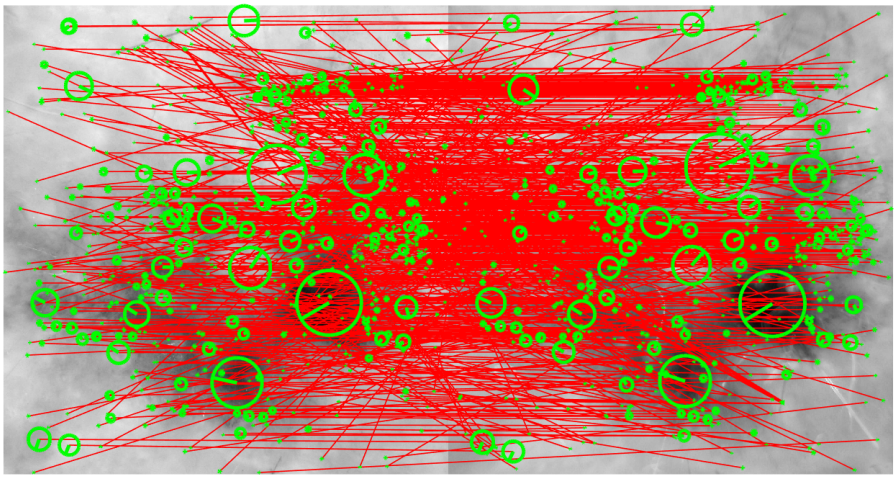


Figure 2.8: Matches between the two skin images using algorithm of VLFeat.

The figure 2.8 shows the matches between the two retina images from the Dataset00, the number of matches is 639. As before, the matches were realized with the default values, the inputs used were obtained from the sift function. From the total amount of key points obtained between the two skin images, less than 2% got a match. Where a fine tuning is needed, because, with a visual examination, outliers can be found in a big quantity.

2.3 Comparison between Lowe's algorithm and third party implementation

The initial difference between those two implementation is the amount of parameters you can change, for the Lowe's implementation few parameters can be changed within the Matlab scripts, because the implementation is based on Lowe's paper, in which he ran several experiments to find the possible best set of parameters to acquire the best descriptors. The only inputs needed are the images for both the script to get the key points and the script to match two images. If the user wants to change something, it has to be realized directly into the scripts with the possibility of changing something which affects the performance. In the match script, the *distRatio* is the only variable which can be easily changed to affect the ratio of vector angles from the nearest to second nearest neighbor is less than distRatio.

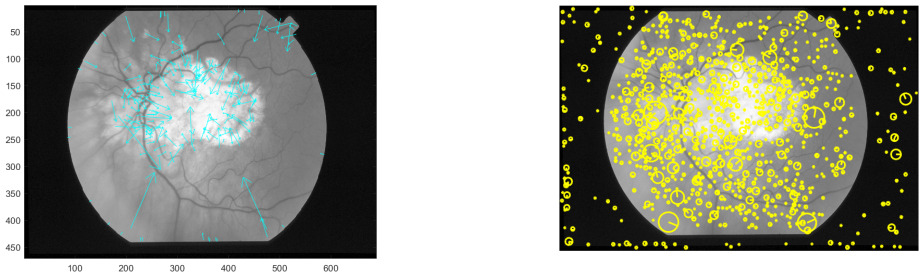
The VLFeat implementation gives the user more possibilities to change parameters, the next list shows the most important optional inputs:

- **PeakThresh, 0:** Set the peak selection threshold.
- **EdgeThresh, 10:** Set the non-edge selection threshold.
- **NormThresh, -inf:** Set the minimum l2-norm of the descriptors before normalization. Descriptors below the threshold are set to zero.
- **Magnif, 3:** The scale of the key point is multiplied by this factor to obtain the width (in pixels) of the spatial bins.
- **Orientations:** If specified, compute the orientations of the frames overriding the orientation specified by the 'Frames' option.

Another difference is at the moment of matching the descriptors, the Lowe's algorithm not only matches the descriptors from two images, but it displays the results with a line connecting the descriptors which match, with an optional output in order to get the number of matches or not.

Matching the descriptors of two images with the implementation of VLFeat only returns the two sets of SIFT descriptors which matched and the squared Euclidean distance between the matches, the last one as an optional output. Meanwhile, the inputs are going to be only the vectors of the descriptors to analyze. The benefit of this implementation is the optional input, it allows to change the threshold in which the descriptors are considered as match or not.
Unlike the Lowe's implementation, a script has to be written separately if the display of the matching points is wanted, as Figure 2.7 and Figure 2.8.

The Figure 2.9 shows the key points using the corresponding implementations, the retina image is used to compare both implementations because of the amount of key points. The most significant difference is the **amount of key points**. The thresholds established along the algorithms are the reason of why in the Figure 9(b) there are more key points than in the Figure 9(a), such as minimum contrast, ratio of principal curvatures, peak threshold and normalization threshold, the last two can be adjust in the VLFeat implementation.



(a) Key points using Lowe's algorithm.

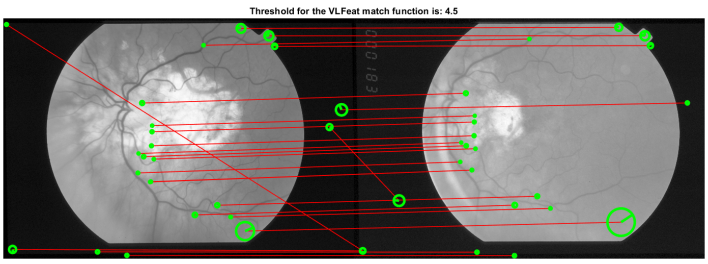
(b) Key points using the VLFeat function.

Figure 2.9: Key points shown using both implementations.

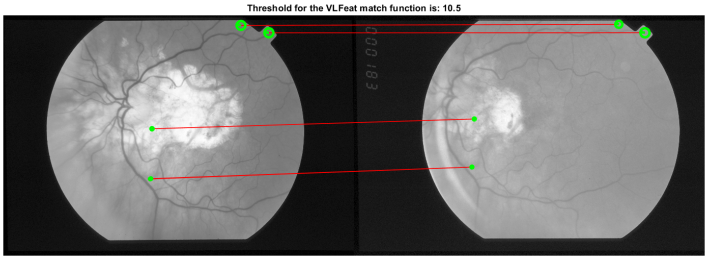
In order to replicate the results from Lowe's implementation with the third party implementation is needed to find the correct values for the different thresholds. It might be necessary to do some very fine tuning on other parameters if the purpose is to fully replicate the results.

2.4 Experiments with different parameters

The Figure 2.10 represents the results obtained after changing the threshold on the matching function of the third party implementation with the purpose of replicate the results from Lowe's implementation. The Figure 10(b) shows the closest result to Lowe's implementation in which the threshold is set up to 10.5. The result is very similar to the Figure 2.3, the difference is all the matching descriptors are within the retina and the points in the center of the retina are similar to the Lowe's implementation, but not the same.



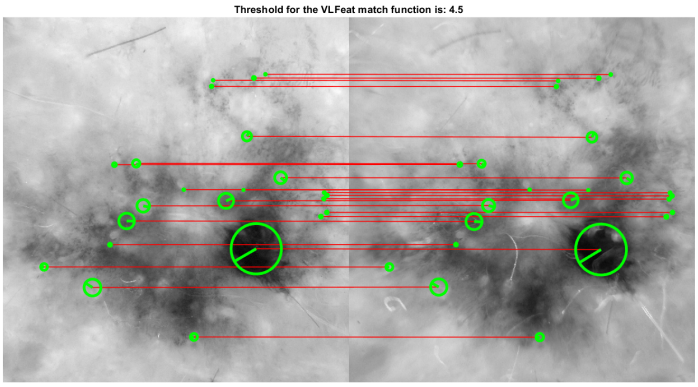
(a) Matching descriptors with 4.5 as threshold.



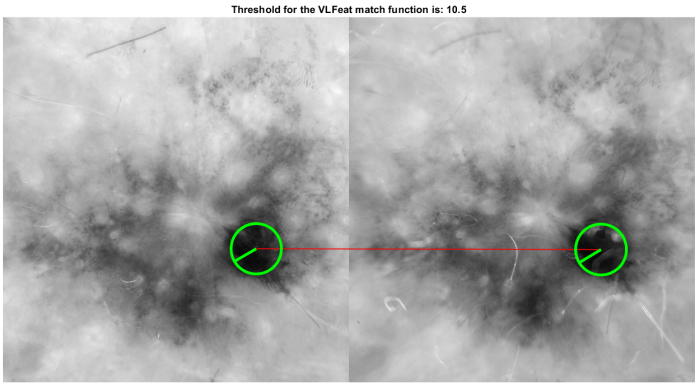
(b) Matching descriptors with 10.5 as threshold.

Figure 2.10: Experiment with different threshold on the VLFeat match function.

The Figure 2.11 represents the results obtained after changing the threshold on the matching function of the third party implementation with the purpose of replicate the results from Lowe's implementation. The Figure 11(a) shows the closest result to Lowe's implementation in which the threshold is set up to 4.5. The result is not only very similar to the Figure 2.4, but at the same time the remaining outliers were deleted.



(a) Matching descriptors with 4.5 as threshold.



(b) Matching descriptors with 10.5 as threshold.

Figure 2.11: Experiment with different threshold on the VLFeat match function.

As the Figures 2.10 and 2.11 show, the results from the third party implementation can be very similar to the Lowe's implementation by only changing the threshold on the match function. That is why the VLFeat implementation is more flexible with the parameters, so the user can adjust it depending on the final objective. Because, using as reference the experiments ran, it does not mean the algorithm is going to work always if the threshold is the same for different images. The threshold also helps to get rid of some outliers, the main thing is to define the objective and to try different values for the parameters to achieve the best possible result.

The thresholds have to be adjusted in the sift function for replicating the results where only the key points are shown. In mentioned function, the thresholds are for the edges, peaks, as mentioned in previous sections, the third party implementation allows the user to change values according to the final purpose.

3 Register Image Pairs Estimating Homography

Once a set of features have been detected on both images of a given pair, those can be associated according to the similarity of their descriptors in what is known as the matching step. This association will mainly rely on computing the distance between the descriptor vectors of each feature to determine the best candidate to represent the same feature in both images.

When this association has been done, the location of these feature pairs (or matchings) can be used to estimate a homography matrix, describing the transformation of the plane containing them between the two views.

In the Homography2.py tasks concerning Part 2 of the Lab are automated. First, the images are automatically loaded in 2 loops for homography computation, registration and metric calculation. The function that governs the computation of homographies consists in 2 types of algorithms, 1 for the euclidean and similarity cases and another for the affine and projection. This approach was used after performance testing with both mindsets an it would be best to present both approaches regardless to present the fact that while the method may differ, homography can be computed in a number of ways if the concept behind the implementation stays the same: both approaches solve the same system of equations presented in the slides. The first is very python-esque but doesn't resemble the matrix layout presented in class, the second one is more simplistic but it is closer to what was presented during the lectures. The script contains two keypoint extraction strategies, 1 starting from the Features.mat provided by the professor and the second one with automatic feature detection and computation provided by the OpenCV framework that was used for the python part of this project. The script also contains functions for drawing correspondences, image overlaying and input-output routines.

Once the function with the following shape **H = computeHomography(Features, Matches, Model)** was computed, the features coordinates stored in the **Textures.mat** file were used to compute the homografies between the reference images **00.png** and the other three.

(a) Overlay between reference and image 2.

(b) Overlay between reference and image 3.

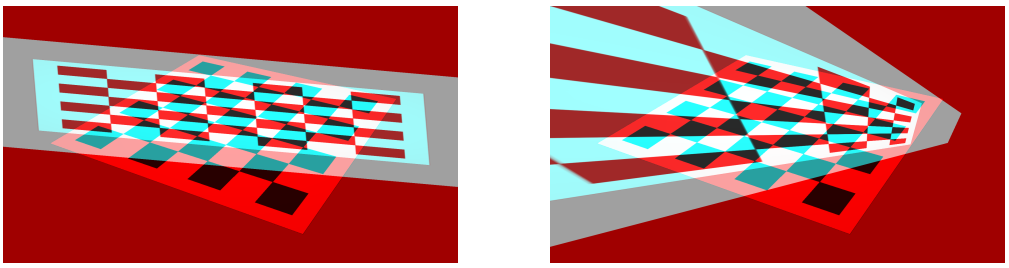
Figure 3.1: Overlay of images using similarity model.

The model which fits better the planar transformation between the reference image and the rest is **similarity**. It can not be consider as the best possible result, but comparing to the other models is the one with the best performance.

The Figure 3.1 shows both overlays, between the reference image and the second image, and between the reference image and the third image from the data set. The purpose of this is to

show how by using the same model there can be a good result, Figure 1(a), or a bad result, Figure 1(b).

Using motion models with more degrees of freedom than the theoretically needed in some image pairs is consider to be a drawback, because the algorithm exceeds on its performance. In the Figure 3.2 shows the results obtained using different models, affine and projection, between the reference image and the image 3 from the data set. The model with best result is the one with less degrees of freedom, affine.



(a) Overlay using affine model.

(b) Overlay using projection model.

Figure 3.2: Overlay between reference and image 3

The table 3.1 shows the values obtained in order to evaluate the performance of the algorithm without RANSAC. To get those values, the procedure is to apply H to the points detected on the source image and measure their euclidean distance to their matches in the destination image. The next step is to apply the inverse of H to the points detected on the destination image and measure their euclidean distance to their matches in the source image.
The first number for each of the cells in the table represents the first step and the second number represents the second step, both mentioned previously.

Table 3.1: Distance between matches for Dataset01.

Applying H and its inverse to measure distance between matches				
	Affine	Euclidean	Similarity	Projection
Image 1	26.2803/26.1143	26.2803/26.1143	26.2803/26.1143	26.2803/26.1143
Image 2	30.3803/29.6528	30.3804/29.6528	30.3803/29.6528	30.3803/29.6528
Image 3	19.0136/17.1092	19.0136/17.1092	19.0136/17.1092	19.0136/17.1092

The homography matrices obtained using the correspondences suggested by the match.m function allow a better registration, not completely accurate, nor with significant misalignments. The visual evaluation shows the results are good, since the function provides several correspondences to do the registration. The Figure 3.3 shows the overlay of both images, the source and destination image, once the registration is done.

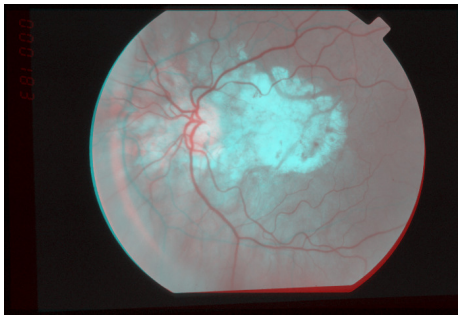


Figure 3.3: Overlay of retinas images from Dataset00.

The Figure 3.4 shows the overlay of both images, the source and destination image, once the registration is done for the skin images from the Dataset00. Which shows how the images pairs are paired with a good level of accuracy.

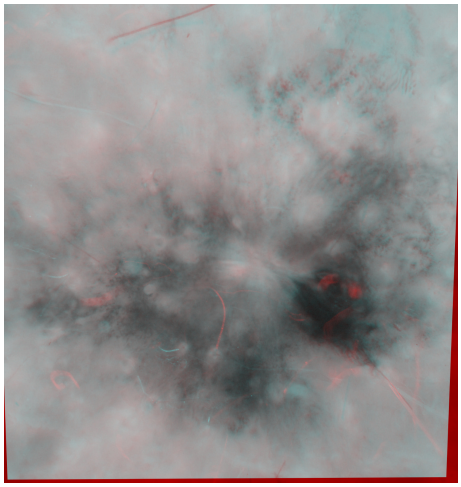
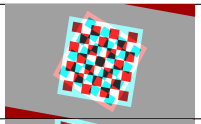
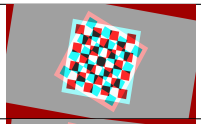
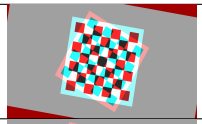
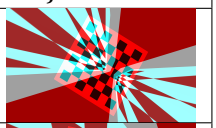
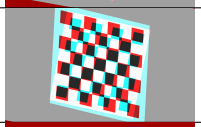
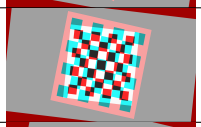
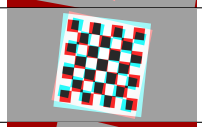
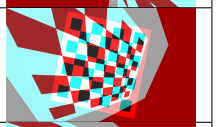
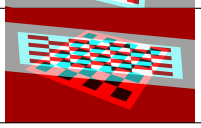
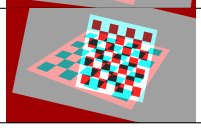
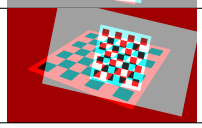
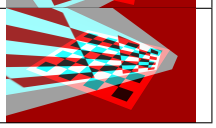


Figure 3.4: Overlay of skin images from Dataset00.

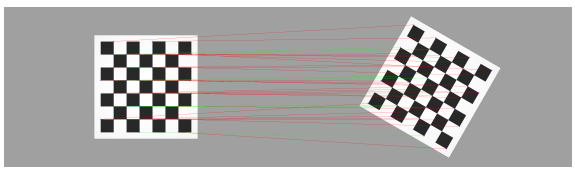
The Table 3.2 shows the results obtained for this part of the laboratory activity, which the differences can be seen clearly depending on the model used. For some cases, the model with less degrees of freedom is a better option than the one with more degrees of freedom.

Table 3.2: Overlay between reference image and data set with all models.

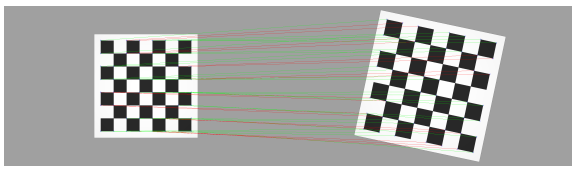
Overlay of images				
	Affine	Euclidean	Similarity	Projection
Image 1				
Image 2				
Image 3				

4 Improving the Registration Accuracy

Homography3.py inherits the same architecture as Homography2.py but wraps the homography finding part of the code in a RANSAC routine. Because of this, all the code was slightly modified to incorporate new information such as inlier representation for the drawing functions.
In both scripts, the reprojection error was firstly coded in the way suggested in the Lab but an unidentified bug was rendering it useless. After presenting our findings and errors to the professors, together with him, we devised a new algorithm that is based on the concept of the reprojection error and it can be found in all the scripts that output this metric.



(a) Inliers and outliers between reference and first image.



(b) Inliers and outliers between reference and second image.

Figure 4.1: Inliers and outliers using similarity mode

The results obtained with the RANSAC-based homography estimation are more accurate than the previously obtained, because the algorithm helps to get rid of the outliers and allows the algorithm to work with the best matches possible. In the Figure 4.1, the inliers and outliers can be visualized with green and red lines, respectively. The outliers at the end are eliminated so the algorithm can work with the inliers, the more amount of inliers the better the result it gives.

The use of data normalization helps reducing the euclidean distances between the matches of the images because the information is processed in one place and only in one place, thus reducing the probability of inconsistent information.
The table 4.1, as the table 3.1, shows the distance between matches with RANSAC for all the models and all the registrations, the reference image with all the other three images. The first number of the cells represent the euclidean distance between the matches of the source image and the destination image; the second number represents the distance, but applying the inverse of H to the points detected on the destination image and the distance to their matches in the source image.

Table 4.1: Distance between matches for Dataset01, applying RANSAC.

Applying H and its inverse to measure distance between matches without normalization				
	Affine	Euclidean	Similarity	Projection
Image 1	15.0004/14.8975	26.2741/26.3508	26.6284/26.3508	26.2803/26.3508
Image 2	29.1458/28.9891	30.3524/29.5786	30.3524/29.5787	30.6810/29.8420
Image 3	18.8675/17.9231	19.5342/18.1564	19.1029/18.4230	19.1586/17.5620

The Table 4.2 represents the values distances between matches of the source image and the destination image. The framework used for this activity was Python and the programming library OpenCV which provides the final result with an automatic process, for both normalization and transformation. Due to that, the Table 4.2 shows only one cell for each one of the images without the specific model used to achieve the final result, because the library finds the best possible model with a step to normalize the data.

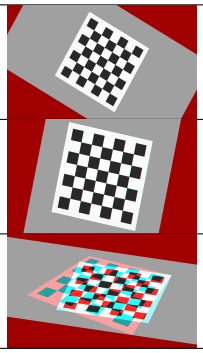
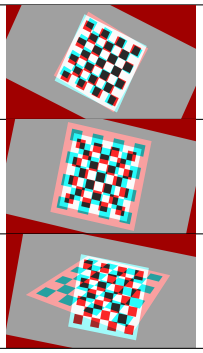
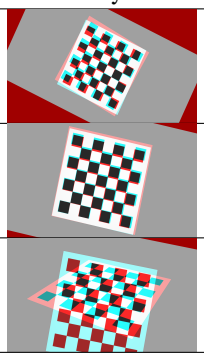
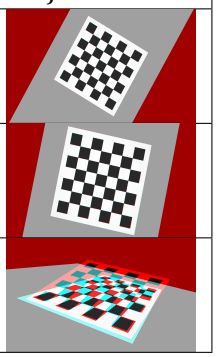
Table 4.2: Distance between matches for Dataset01, applying RANSAC and normalization.

Distance between matches with normalization	
	Transformation
Image 1	31.3169/33.0860
Image 2	31.0922/31.1067
Image 3	11.7389/11.6725

The Table 4.3 shows all the results obtained implementing the algorithm with RANSAC, with deletes the outliers. Where if the Table 3.2 and Table 4.3 are compared, the results show how implementing RANSAC the performance is better. The model which represents better the difference is the *projection model*, where for the image 1 and image 2 the results are accurate if they are compared to same results obtained without RANSAC.

The final images obtained during several experiments were different in some cases. The difference lies in the direction of the image itself, since the image can be rotated 90° and it remains with the same visualization of the chessboard.

Table 4.3: Overlay between reference image and data set with all models, using RANSAC.

Overlay of images				
	Affine	Euclidean	Similarity	Projection
Image 1				
Image 2				
Image 3				