Universitat de Girona

MAIA

UNIVERSITAT DE GIRONA, MEDICAL IMAGE REGISTRATION AND APPLICATIONS

Report: Lab 1 | 2021-2022

OJEDA, Manuel; VASILE, Alexandru

October 14, 2021

1 REGISTRATION FRAMEWORK AND COMPONENTS

For the framework used, the principal components to be chosen are:

- **Unregistered image** also know as "Moving image".
- **Template image:** also know as "Fixed image".
- **Metric type:** with the possibility of choose between sum of squared distances, gradient correlation and cross-correlation.
- **Type of rigid transformation:** with the possibility of choose between rigid and affine.

The type of the images are gray scale intensity image with double precision and the declaration of them are found at the beginning of the *main file*. The type of metric is found in the line 9 of the *affineReg2D* function before the type of rigid transformation.

In the same function, the initial parameters are set for the type of transformation, from line 14 to 22. The values are initialized for both rigid and affine transformation, for both options the scale vector is declared and its function is to rescale the image according to the value of each of the elements in the vector for X, Y, and Z. The values for the affine transformation matrix were initialized based on keeping the original image without any change and start from that point the process, that is why the vector for the affine option are either none or very small at the beginning.

From line 29 to 39, the multi-resolution step can be found; the levels used for this framework are 6 and the resized images are saved in an array for later use for the calculation of the minimum error of the registration process. If image is M-by-N, the size of the output is ceil(M/2)-by-ceil(N/2).

The optimization stage can be found between line 44 and 56; it minimizes the error with the metric chosen at the beginning, implementing the function *affine registration*.

The function mentioned uses the affine transformation of the (3D) input multiresolution stack and calculates the registration error after each transformation. The inputs are:

- **Parameters in 2D:** if the transformation is rigid, the vector contains *translation in X, translation in Y, and rotation*; else the vector contains *translation in X, translation in Y, rotation, resize in X, resize in Y, shear in XY, and shear in YX.*
- **Scale:** Vector with scaling of the input parameters with the same length as the parameter vector.
- **Image moving:** image which is affinely transformed.
- **Image fixed:** image which is used to calculate the registration error.
- **Metric:** same metric defined at the beginning of the algorithm.
- **Transformation type:** same transformation defined at the beginning of the algorithm.

First, the parameters and the scaling factors are initialized based on the type of transformation which will be used to transform the image. The function *affine transform 2d double* is in charge of getting the image transformed based on the matrix and the kind of interpolation selected. The result is used to calculate the error between this image and the fixed image, with the selected metric. The formula to calculate the error using squared differences is in the line 59, the equation for the gradient cross correlation is from line 63 to 70, and the equation for the cross correlation is in line 97.

The same optimizer used in this stage is in charge of displaying the plots where we can see how the error is decreasing at every iteration and to display the values in the command window, where the most important are: *error and procedure applied to the image.*

With the minimum possible error, from line 60 to 82, the image is processed again with the best possible parameters obtained from the algorithm to get the final result. Same results which are presented at the plots for a visual validation.
From line 88 to 95, at the end of the algorithm, the final information for: *translation in X, translation in Y* is shown in the command window.
The last lines, from 98 to 102, are to show the final results with the images in a subplot 4-by-4, where the fixed image, the moving image, the transformed image and the result of the subtraction between the image transformed and the fixed image.

2 Implementation of multiresolution

This stage is implemented between the lines 29 and 39, the number of levels are 7, where the first one is the original size. All the images with different resolutions are kept in an array which is checked in the next steps to realize the different steps for the registration to work. It is a loop from the first level to the last one defined where the function named *Multiresolution* is called, the input is M-by-N and the size of the output is ceil(M/2)-by-ceil(N/2).

The next step of the multiresolution process is found in the optimization stage, in this stage the process mentioned above is repeated for all the images with different resolutions. To achieve this, the initial values of the vector of parameters are updated at each iteration, by doubling the value for translation in X and in Y, the other values are kept constant.

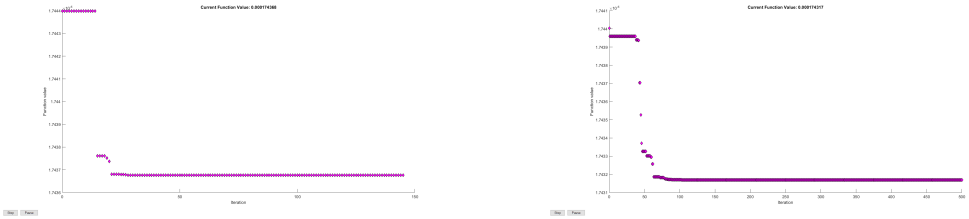
3 Results and discussion

The figure 3.1 shows the first implementation with full algorithm. For both of the figures, the two images found on the first row are the fixed image and the moving. In the second row, for each one of the figures, the images are the image registered and the result of the difference. For the experiment with those images, the metric used is squared differences and the transformation type is rigid.

(a) Registration with rigid transformation. (b) Registration with affine transformation.

Figure 3.1: Registration with the metric squared differences

In figure 3.2 the plot of error value can be seen for each one of the iterations; in this case the maximum number of iterations are 500. In the beginning a high error can be observed and then it decreases with each iteration.



(a) Plot of error with rigid transformation. (b) Plot of error with affine transformation.

Figure 3.2: Plot of the error with the squared differences metric

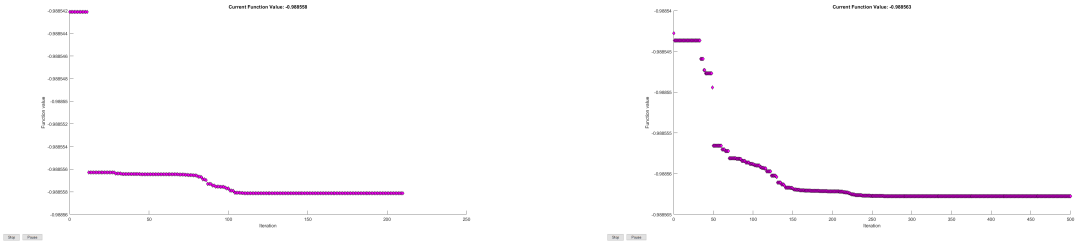
The figure 3.5 shows the results implementing the gradient cross-correlation metric. As before, the two images found on the first rows are the fixed image and the moving. In the second rows, the images are the image registered and the result of the difference.

(a) Registration with rigid transformation. (b) Registration with affine transformation.

Figure 3.3: Registration with gradient cross-correlation metric

The figure 3.6 shows the plot of error value for each one of the iterations, which in this case the

maximum number of iterations are 500. Having at the beginning a high error and then having a very small error.



(a) Plot of error with rigid transformation.

(b) Plot of error with affine transformation.

Figure 3.4: Plot of the error with gradient cross-correlation metric

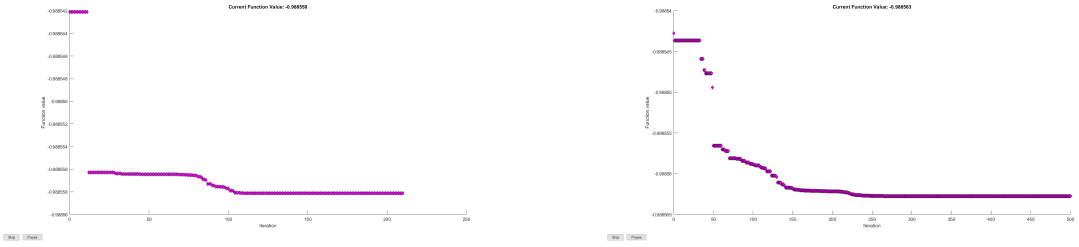
The figure 3.5 shows the results implementing the gradient cross-correlation metric. As before, the two images found on the first rows are the fixed image and the moving. In the second rows, the images are the image registered and the result of the difference.

(a) Registration with rigid transformation.

(b) Registration with affine transformation.

Figure 3.5: Registration with gradient cross-correlation metric

The figure 3.6 shows the plot of error value for each one of the iterations, which in this case the maximum number of iterations are 500. Having at the beginning a high error and then having a very small error.



(a) Plot of error with rigid transformation.

(b) Plot of error with affine transformation.

Figure 3.6: Plot of the error with gradient cross-correlation metric

The figure 3.7 shows the results obtained by implementing the squared distances in figure 7(a) with rigid transformation and gradient cross-correlation metric in figure 7(b) with affine transformation. The difference are in the images used for the algorithm, the images used are named

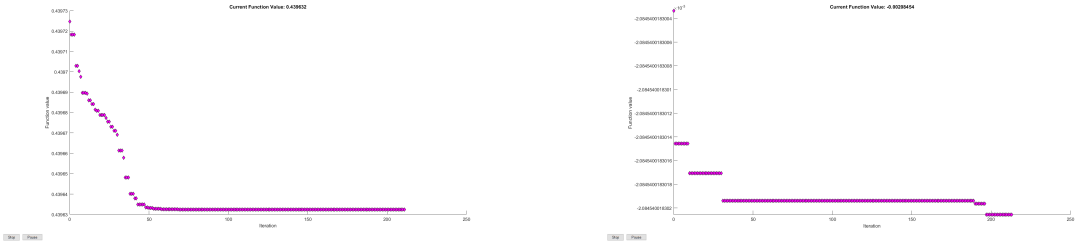
"brain3.png" and *"brain4.png"*. It can be observed the fixed and moving images are similar, the difference are in how the values are inversed (it can be observed in the background) which this affects the algorithm to work correctly. It can be corrected by adding a stage of pre-processing where it can be assured the images are not going to be inverted.

(a) Registration with rigid transformation.

(b) Registration with affine transformation.

Figure 3.7: Registration with squared distances and gradient cross-correlation metric

The figure 3.8 shows the plot of error value for each one of the iterations for the previous processed images. Based on the metrics used, it can be seen that the error is big because of the incongruity with the result obtained.



(a) Plot of error with rigid transformation.

(b) Plot of error with affine transformation.

Figure 3.8: Plot of the error with squared distances and gradient cross-correlation metric

In table 3.1, the values of the best metric and its computational time are mentioned according to each one of the parameters. It can be observed the one which took the less amount of time was the cross-correlation with rigid transformation for the first configuration of images, moving image as "Brain1" and fixed image as "Brain2". While the one which took the most amount of time was gradient cross-correlation metric with affine transformation.

Table 3.1: Values of metric according to the metric and transformation and computational time.

Value of metric and computational time (seconds)						
	sd-r/time	sd-a/time	gcc-r/time	gcc-a/time	cc-r/time	cc-a/time
Brain1 vs Brain2	0.9886/56.44	0.9886/117.6	0.9886/91.8	0.9886/141	0.9711/45.2	0.9724/103.8
Brain3 vs Brain4	0.021/50.01	-0.0004/108.3	0.009/94.24	0.038/108.55	-0.992/39.86	-0.008/104.75

4 Report of problems encountered during the assignment

The first problem that was encountered in this project was that such a framework is designed around the idea of multiresolution. Because of this it was believed that results obtained before the implementation of said idea were wrong because of erroneous implementation of the metric functions.

Another challenge was the design of the multiresolution approach. Mere subsampling lead to poor results, hence, a *Gaussian* blur had to be implemented to improve the performance of this technique. This was an iterative process but it would be too long to present it in here.

The third and final problem was the one of the images with inverted values, but as was stated, it was quickly fixed by some simple pre-processing.