Medical Image Registration Lab Report

Raabid Hussain

April 29, 2016

1 Objective

The objective of this lab work was to get familiarized with the basic structure of all the image registration algorithms. A sample code was provided in MATLAB for better understanding of the registration framework. We had to test the code and modify a few steps in order to improve or widen the scope of the code.

2 Introduction

Image registration has become a popular subject in medical imaging. It can be defined as the process of aligning two or more images of the same scene, such that they are a perfect fit. With the introduction of different medical imaging techniques, it was found that whenever multiple images need to be taken of the patient in the same pose, the images would never match because it was very difficult to keep the patient in the exact spot. So a need for image registration frameworks arose. It is mostly used as a pre-processing steps for image analysis.

Image registration involves assigning one the image as the reference, commonly known as fixed image. The other image, commonly called the moving image, undergoes a series of geometric transformations until the image gives the best possible match with the fixed image. The transformation maps location in one image to new locations in another image. The key to a perfect registration is to determine the correct geometric transformation parameters.

3 Image Registration framework

A typical registration framework consists of 4 main components, namely a similarity index finder, a optimizer function, a geometric transformation and a linear interpolator. A flow diagram of the framework is shown in figure 1. A code was provided to us with each of these blocks pre-written and we had to improve or add on that code. The original code was based on D. Kroon's work in University of Twente.

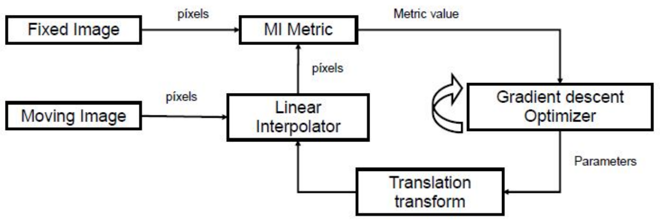

Figure 1: Flowchart of a typical image registration framework.

The code, provided, is split into four different script files. The main one is called 'affineReg2D.m'. This is the file that takes the fixed and the moving image as inputs and calls each of the components of the registration framework. All the parameters for the code are also set in this file. The main parameters are the type of similarity score we want to match (the code for sum of squared distances was already implemented and we had to implement the mutual information metric), the type of registration (rigid transformation was already implemented and we had to implement the affine transformation), and should the registration be done in multi-resolution or not (we had to include this variable and implement the multi-resolution registration framework).

First a gaussian filter is applied onto both of the input images to remove noise and blur the image. This is done so because the spatial information can never be exact and this no two regions can be precisely registered. So, a gaussian filter is applied as it has a spreading out effect on different regions, thus reducing discrepancy and increasing chances for a successful registration.

A big problem with this framework is that it requires an initial estimate of the transformation. This causes a problem because if this initial estimate is far from the a actual transformation, the algorithm gets stuck in the local minima or maxima. For this later a multi-resolution framework will be implemented to speed up the process by providing an estimate of the transformation using lower resolution images. The initial parameters are first scaled according to the image being used. Like when doing multi-resolution an initial estimate is obtained from a lower resolution image. When we have to shift to the higher resolution image, all the parameters except the translation ones can be kept same. Since when we scale the image to double the size, the translation parameters also have to be multiplied by two to keep in the same scale.Also another scaling parameter is introduced into the framework. This can be adjusted to give more significance to some of the parameters.

After this, the inputs are fed into an optimizer function called the 'fminsearch'. Although this ia a very popular minimizer function in MATLAB but it is not optimum. It is common for it to get stuck in local minima and not yield the minima we want. For this optimizer, a cost function has to be input too. This is done through the 'affine_function.m' file. This file applies the transformation being tested on the moving image and uses this transformed image to get the value of the cost function. In our case, the sum of squared distances was already implemented in this function file and the rigid transformation is used. Rigid transformation uses three parameters: translation in x, translation in y and rotation. The center of the rotation is on the orthogonal z-axis.

After the optimizer has reached a solution, the transformation is applied onto the image to get the final output. For this two function files are used, namely 'affine_transform_2d_double.m' and 'image_interpolation.m'. The main problem with transformations is that when they are applied on the input image to get the final image, there are many pixels that are left blank and black patches are observed on the final image. To solve this, reverse transformation and interpolation have been implemented through these two functions. So now, an inverse transformation and type of interpolation is determined depending on the output of the optimizer. This inverse transformation is used to get the value of each pixel on the output image form the transformed image.

A sample output of the program, without any change, is shown in figure 2:

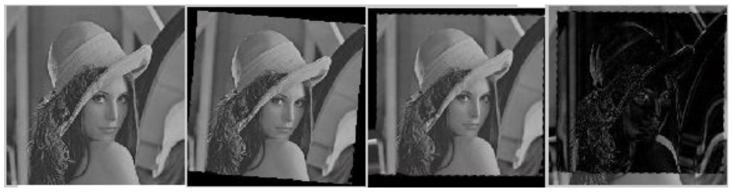

Figure 2: Output of the program using rigid transformation and sum of squared distances. LR (fixed, moving, output, difference images).

4 Similarity Metric

In the framework provided, the sum of squared distance was being used. However, this does not lead to optimum results. So a new similarity metric called the mutual information was implemented. It uses entropies of the two images to get the similarity index. Lower entropy values indicate higher similarity between the images. The following equation defines the mutual information:

$$C(A,B) = \sum_{i=0}^{I} \sum_{j=0}^{J} p_{AB}(i,j) \log \frac{p_{AB}(i,j)}{p_A(i) p_B(j)}$$

$$= \quad H(A) + H(B) - H(A,B)$$

In order to calculate the mutual information, first histograms of both the images was computed. 'imhist' function of MATLAB was used for this. This was followed by computing a 2D histogram of the two images together. This histogram indicates the likeliness of a pixel to have a specific value in image 1 and a specific value in image 2. The above equations were implemented to get the metric value. However, since we are using an optimizer which settles on the minima and we need to compute the maxima for mutual information metric so, the negative of this metric value was used as input to optimizer system. These modification is more preferable for image with a change in color descriptions. A sample result is shown in figure 3.

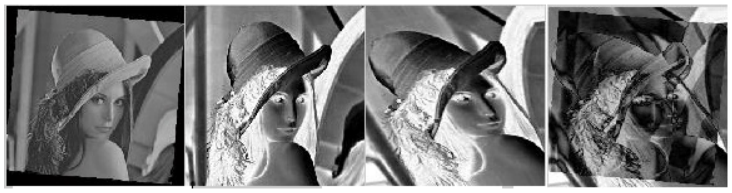

Figure 3: Output using affine transformation registration framework. It uses affine transformation and mutual information registration framework. The scale was chosen to be [100 100 1 100 100 1]

5 Transformation

Rigid transformation was already implemented in the code provided. However, rigid transformation only takes into account translations in x and y axis and the rotation of the image. It does not deal with scaling and shearing. This makes the algorithm lesser efficient and more prone to error as the input moving may have different characteristic changes. In order to include more information, affine transformation was incorporated into the existing code. Affine transformation takes into account additional features. The transformation is now governed by the following equation:

$$\begin{aligned} f_x(x,y) &= a_x x + a_y y + t_x \\ f_y(x,y) &= b_x x + b_y y + t_y \end{aligned}$$

This allowed for more complex images to be input into the system. The transformation matrix previously implemented was changed and a different parameter for all its top six elements was declared. For better results the parameters governing the image scale were set to 1 initially instead of 0 (for all other parameters). Since affine takes care of six parameters to be optimized rather than three parameters,sometimes the result was not as good owing much due to constraints of the optimizer and the maximum iterations possible. A sample output is shown in figure 4:

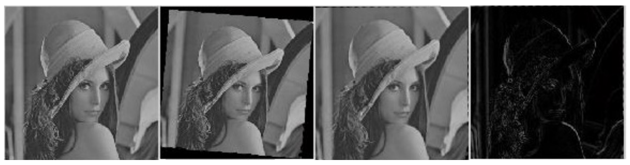

Figure 4: Output using affine transformation registration framework. It uses affine transformation and sum of squared difference registration framework.

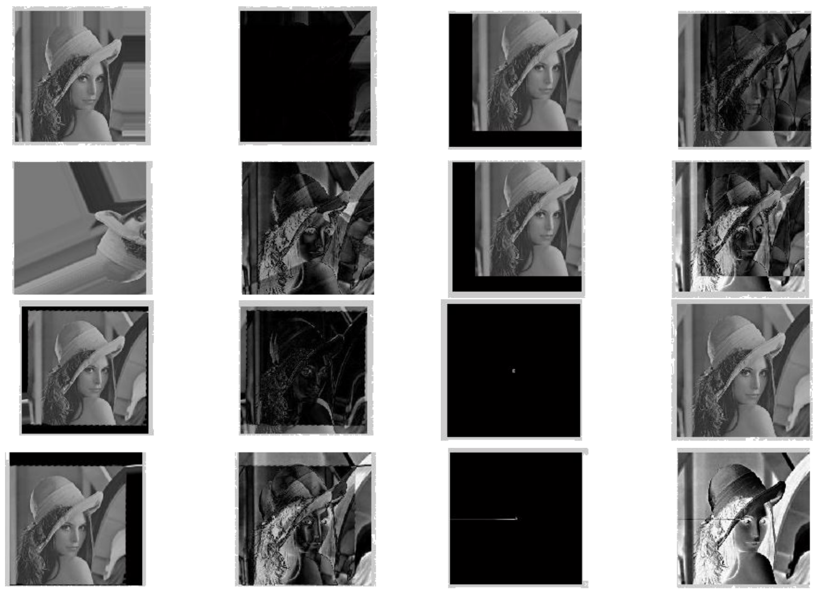

Figure 5: Selected outputs with selected combinations of framework parameters. Left (SSD & rigid), Right (mutual & affine) Top to Bottom Images(1&3, 1&4, 3&2, 3&4)Similarity metric vs number of iterations for two stage resolution registration.

6 Multi-resolution Registration

It is very common that if the initial estimate is far from the actual transformation, the algorithm converges to a completely different extrema. To solve, this multi-resolution framework was introduced. The input image is decreased in size. The registration is implemented on these small images. The output of this registration is then used as an initial estimate for registration on a larger image. As the smaller the image and the closer the estimate to the minima, the faster is the convergence of the algorithm.

However, in this case the output of the smaller image registration can not be directly input to the larger image registration. This is due to the translation parameters. As the image increases in size, the translation also increases with the same amount. So the smaller image output is first scaled appropriately and then fed into the larger image registration framework. Since the size of the image varies as we go down the multi-resolution pyramid structure, so a for loop could not be written and hence only maximum 3 stage pyramids are allowed in the code. A sample output is shown in figure 6. Multi resolution registration also decreases the runtime as the initial registration provides with a reasonable estimate of the registration.

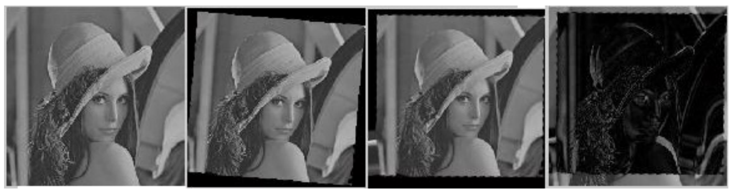

Figure 6: Output using multi-resolution framework. The same images as shown in figure 2 were used. It uses rigid transformation, sum of squared difference and 3 stage multi resolution registration framework.

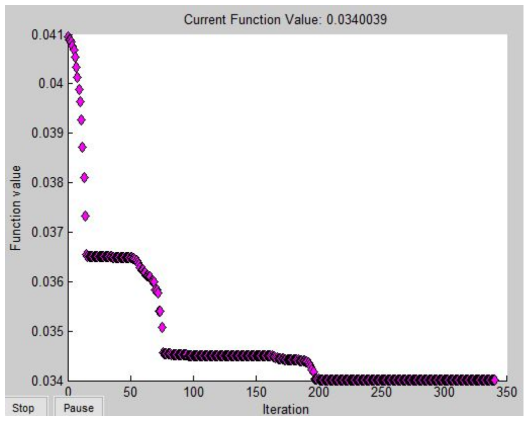

Figure 7: Similarity metric vs number of iterations for single stage resolution registration.

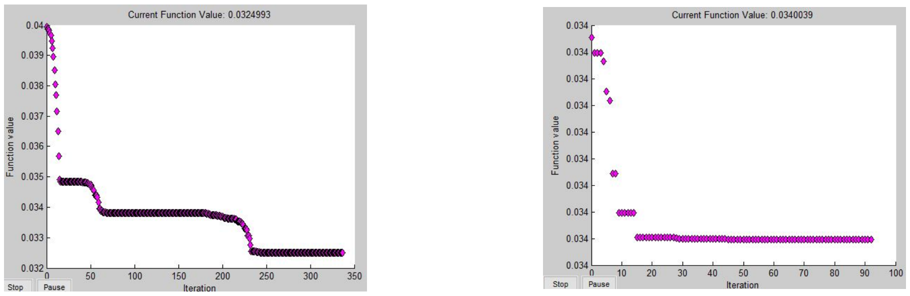

Figure 8: Similarity metric vs number of iterations for two stage resolution registration.

The above graph represent the number of iterations used to reach the goal transformation parameters. Although the total number of iterations are a little more in multi resolution (combined number of iterations for all stages), the ru time is smaller because most of these iterations are on a smaller image, reducing the runtime. Likewise, here a significant decrease in runtime was observed despite similar output. The time with single stage registration was found to be 13 seconds whereas for a two stage registration was 9 seconds and for a three stage registration, the runtime was found to be 6 seconds. The mean square error for all the different staged registrations was found to be 0.0358. One thing to notice was that despite the time reduced enormously, however when the results were analyzed both qualitatively and quantitatively, they were the same for all image combinations with a very slight improvement in a couple of scenarios.

7 Conclusion

In this lab work, a typical framework for image registration was implemented. We were provided with an already running code and we were to modify it to deal with affine transformations, mutual information and multi-resolution registrations. The performance of this exact framework was limited by the use of a simple and general optimizer. The results for different cases were discussed. In general, mutual information metric worked better with affine transformation, whereas SSD worked better with rigid transformation. Also, mutual information showed better results when 1 image is inverted (negative) as it uses entropy rather than the difference.