
Image registration using the Morphon algorithm: an ITK implementation

Release 0.00

Jérôme Plumet¹, Mats Andersson², Guillaume Janssens¹, Jonathan Orban de Xivry¹, Hans Knutsson² and Benoît Macq¹
jerome.plumat@uclouvain.be

February 27, 2009

¹Laboratoire de Télécommunication et Télédetection (TELE), Université Catholique de Louvain, Belgium

²Department of Biomedical Engineering, Linköping University, Sweden

Abstract

Medical image registration is becoming a more and more useful component of a large number of applications. The presented method aims to enrich the ITK library. This method, called Morphon registration algorithm, computes a dense deformation field accepting inputs from different intensity contrasts. This article presents its implementation within the Insight Toolkit.

In this paper, we provide a brief description of the algorithm, a presentation of the implementation, the justification of our modified classes and the results given by the algorithm. We demonstrate the algorithm in application of different images intensity contrasts and dimensions.

Latest version available at the [Insight Journal](http://hdl.handle.net/1926/1338) [<http://hdl.handle.net/1926/1338>]
Distributed under [Creative Commons Attribution License](#)

Contents

1	Introduction	2
2	The Morphon method	3
2.1	The deformation field	3
2.2	The certainty field	4
2.3	The smoothing	4
3	The implementation	5
3.1	Introduction - How to launch	5
3.2	The source code and its structure	5
	The main file	5
	The complex numbers	6

Matrices Operation and iterations' initialization	7
The filters and the convolutions	7
Computation of the deformation field	8
The smoothing	8
3.3 The 2D and the 3D cases	8
4 Results	8
5 Future work	9
6 Conclusion	9
7 Remark	9
8 Acknowledgements	11
9 Appendix	11
9.1 class diagram	11

1 Introduction

The goal of image registration is to geometrically align two images. It aims to find a point to point transformation from one image (called moving or prototype image, named I_M in this paper) to the target image (called fixed or input image: I_F). Many image registration methods were developed in the past years. The ITK library already includes some non-rigid image registration algorithms like Demons [3, 7], Block Matching [8], B-Spline [1] or Correspondence-Based Software Toolkit [10]. But computing dense non-rigid deformation fields from images with different intensity contrasts is still a tough challenge. The main advantages of the Morphon algorithm are its sub-pixel precision and its tolerance to gray levels variations in the two images but also in one image.

The Morphon algorithm started being developed 4 years ago and was presented for the first time in [6]. One of the particularities of this algorithm is its metric: it computes the local phase for each voxel of the image in order to perform the displacement estimation. Another particularity of this method is the use of a certainty matrix in order to add a weighting factor to the regularization of the computed displacement field.

This paper is divided in three parts. First, a brief description of the Morphon algorithm is given. Then, the implementation is explained and discussed. Finally, results given by the ITK implementation of the Morphon are presented.

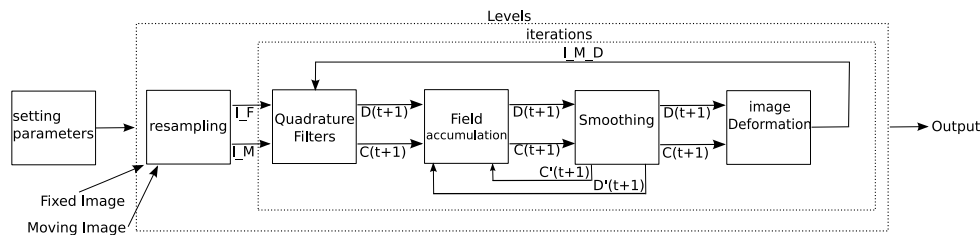


Figure 1: The Morphon's pipeline scheme.

2 The Morphon method

The Morphon method is a non-rigid and dense deformation field registration method developed at the Linköping University in Sweden. This section only gives a brief overview of the algorithm more details can be found in [6, 11].

The images features used to build the dense deformation field (\mathbf{d}) are the information found after filtering the images with directional quadrature filters. The characteristics given by the quadrature filters are the local phase. The directional local phases computed at each voxel are used for two purposes: an estimation of the direction and magnitude of the local displacement and a certainty matrix (C) derived from the magnitude of the filter response and used for field accumulation and regularization.

The Morphon algorithm has a multiresolution approach: at each resolution level, the images, the certainty matrix and the deformation field are resampled to have a specific size, fixed by a scale parameter. The field and certainty matrix are iteratively updated in order to achieve convergence at each resolution step. For each iteration, a sequence of operations (illustrated in Figure 1) is executed. The most important of them are:

- filtering of the deformed I_M ,
- computation of \mathbf{d}_l and C_l , the increment for the field and certainty,
- computation of \mathbf{d}'_a and C'_a , the accumulated field and certainty based on the increment \mathbf{d}_l and C_l , and on the previously accumulated field and certainty \mathbf{d}_a and C_a ,
- regularization of \mathbf{d}'_a and C'_a ,
- deformation of I_M based on \mathbf{d}'_a .

In fact, on the first levels (*i.e.* where images are the smallest) the computed deformations are quite important and have a high influence on higher resolution levels. At the opposite, deformations computed on the last levels have less influence on the global displacement and are used to compute the local components of the deformation field.

2.1 The deformation field

The displacement estimation aims to compute a mathematical transformation which, when it is applied to I_M , produces an image as close as possible to I_F . The displacement is estimated, for each scale, based on conjugate products of quadrature filters which compute the local phase of each image at each voxel. These

estimates are performed in $\frac{D \times (D+1)}{2}$ directions (where D is the number of dimensions in the image, [4]) -in 2D, three filters are sufficient but using four filters usually improves the results. The increment at each iteration, \mathbf{d}_l is estimated as a solution of a least squares problem and C_l is estimated with the amplitude of the filter response.

The field and certainty increments are used to update an accumulated deformation field \mathbf{d}'_a :

$$\mathbf{d}'_a = \frac{\mathbf{d}_a \times C_a + (\mathbf{d}_a + \mathbf{d}_l) \times C_l}{C_a + C_l} \quad (1)$$

Notice that the multiplication (defined with the operator \times here) refers to “point to point” scalar multiplication. Therefore, if a and b are matrices:

$$(a \times b)(\mathbf{x}) := a(\mathbf{x}) \times b(\mathbf{x}) \quad \forall \text{ point } \mathbf{x} \text{ of the images}$$

and if a is a field and b is a matrix

$$(\mathbf{a}_d \times b)(\mathbf{x}) := \mathbf{a}_d(\mathbf{x}) \times b(\mathbf{x}) \quad \forall \text{ dimension } d \text{ and point } \mathbf{x} \text{ of the images}$$

2.2 The certainty field

The response of the quadrature filter is the local phase, thus it measures a local “distance“ between the current voxel and the nearest extreme values. Moreover, edges and lines will have a stronger response than uniform areas. The Morphon method uses the response amplitude as a certainty measure and uses it to calculate accumulated fields.

The way to update the certainty measure is similar to the deformation field update. The algorithm computes a certainty increment C_l at each iteration and accumulates them to the previously accumulated certainty field C_a :

$$C'_a = \frac{C_a^2 + C_l^2}{C_a + C_l} \quad (2)$$

2.3 The smoothing

The Morphon algorithm uses Gaussian smoothing for two main purposes: to avoid aliasing before down-sampling the images and to regularize the certainty matrix at each resolution step and the deformation field at each iteration. For this last purpose, it uses its own smoothing operation in order to take the certainty measure into account: normalized averaging smoothing (see [5]).

This operation can be resumed with the following pseudo-code, where `gauss_smooth` is the Gaussian smoothing operation and \times the “point to point” scalar multiplication:

```
A = deformation_field x certainty;
B = gauss_smooth(A);
C = gauss_smooth(certainty);
new_deformation_field = B/C;
D = certainty x certainty;
E = gauss_smooth(D);
new_certainty = E/C;
```

We can see that the deformations related with a high certainty measure have a high influence on the smoothing of their neighbours. In a similar way, the Gaussian smoothing will smooth the certainty weighted by itself.

3 The implementation

The algorithm has been implemented using the Insight Toolkit. Hence, to use it you need to have the following software: Insight Toolkit 3.6 version or higher and CMake 2.4 version or higher.

To help the users of the source code, some notations and comments were introduced directly in the code.

3.1 Introduction - How to launch

The Morphon's ITK implementation tries to respect the ITK spirit. This method supports images from different modalities as input.

The main global parameters (*i.e.* the fixed, moving and output images, the number of levels, the numbers of iterations and the standard deviations on each one of them) have to be specified at runtime. For example the command

```
$ ./itkMorphon hand_indata.tif hand_prot.tif out.png 10 1.5 10 1.5 10 4.5 10 4.5  
10 4.5 10 2 10 2 10 2 10 2 10 2 10 2 10 2 10 3.5 10 2.5
```

calls the Morphon algorithm with `hand_indata.tif`, `hand_prot.tif` and `out.png` parameters as respectively the fixed, moving and output images. The following numbers are [number_of_iterations, standard_deviation] couples. Thus, in this case, there are 13 levels with a specified number of iterations (10 on each of them in this case) and the standard deviations equal to 1.5, 1.5, 4.5, .. on each one of them. Notice that the standard deviations are given in pixel space and not in real space. The number of levels is specified by the number of [number_of_iterations, standard_deviation] couples.

This application needs, at least, the three first parameters and one couple [number_of_iterations, standard_deviation] to work correctly.

The given package contains the fixed and moving (which are presented further as the “hands“ results) images. You can also read the “README” file given with the package for more information on how to launch the method.

3.2 The source code and its structure

Because a quite large number of classes and libraries are used to launch the program, a “class diagram” is presented in the section 9.1, page 11. This is not a complete diagram. Only useful or modified classes and useful functions or parameters are illustrated in order for it to be printable on a single page and to be more easily read.

The main file

A simple main function is implemented in the `main.cxx` file. This function aims to check the parameters, read the input files, check them and start the pipeline. This function declares the Morphon filter which

is implemented in the `itkMorphonPipe` library. It needs to set the following parameters **before** using the `Update()` `Morphon` filter's command:

- the moving image, set with the command:


```
SetMovingImage( MovingImageType* MovingImage ),
```
- the fixed image:


```
SetFixedImage( FixedImageType* MovingImage ),
```
- the number of levels:


```
SetLevelNumber ( int nlevel ),
```
- the number of iterations on each level:


```
SetIterationsNumber( unsigned int *nofit ),
```
- the standard deviations for each level:


```
StandardDeviations(int *nofdev),
```
- the output image's name (with its extension):


```
SetOutputName( std::string & OutputName ).
```

The `itkMorphonPipe` library contains the largest part of the algorithm, it contains the global level loop in the `GenerateData()` function. This loop is basically formed with the pipeline presented in figure 1. Notice that, in the current version of the `main.cxx` function, the extension of the output image's name must be `'png'`, for the 2D images, or `'mha'` for the 3D images (but it could be easily changed with a few code modifications).

The complex numbers

Intermediate matrices (*i.e.*, given as the result of the convolution of the images with the quadrature filters and used to compute the deformation field and the certainty matrix) contain complex numbers at each voxels: $M(\mathbf{x}) = a + jb$. There are two possibilities to implement such kind of complex matrix M within ITK. First, we can construct two matrices, the first containing the real part and the second containing the imaginary part. But this is not efficient at all because the number of intermediate matrices stored is equal to the number of filters (and there are 4 filters for 2-dimensional images and 6 for 3-dimensional images) and all of them have the same dimensions as the images.

Thus, we decided to adopt the second implementation which consists in defining complex matrices:

```
typedef typename FilterType::ComplexType      ComplexType;
typedef itk::Image< ComplexType, ImageDimension > ComplexMatrixType;
```

Where `ComplexType` is defined in the file used to construct filters and make convolutions (see further for more informations about this point). In this way, we can produce a more efficient algorithm.

Notice that the users don't have to see these intermediate matrices. But it's important to explain this particular point to justify why we had to modify some ITK classes (as explained further).

Matrices Operation and iterations' initialization

As presented in the previous sections, we compute some specific operations to determine matrices and fields. Thoses are implemented as a particular function. Most of them are implemented in the `itkMorphonRegistrationFilter` library. In addition to compute specific matrices operations, this library is used to initialize each iteration (with the `InitializeIteration()` function) and to save the quadrature coefficients (aims to prevent computing them again at each iteration). The `InitializeIteration()` is used to pass those quadrature coefficients to the library which computes the deformation field: `itkComputingMorphonDeformationField`.

The normalized averaging smoothing (presented in section 1) is performed at each iteration in the `itkMorphonRegistrationFilter` library in the `ApplyUpdate` function. The operations could be done by calling the smoothing functions wich are declared in the class's parent: `MorphonToolboxFilter`. This was done because, this last class is templated over all the data (images and deformation field).

The filters and the convolutions

As said, the Morphon algorithm uses quadrature filters to compute the deformation field. This implementation uses $4 \times 9 \times 9$, and $6 \times 9 \times 9 \times 9$ filters respectively for 2D and 3D images but other filters could be used instead. The filters must be constructed only one time (at the very beginning of the algorithm) and passed through the algorithm's functions as a parameter. Filters used by the algorithm are complex filters and their coefficients are declared as double. To construct thoses filters we implemented the `itkBuildingMorphonFilters` library.

By default, this library goes over one of the two files: `2D_QuadPhaseFilter.csv` or `3D_QuadPhaseFilter.csv` which contains respectively the coefficients used for the 2D and 3D.

For the 2D, the l^{th} line of the file contains the l^{th} coefficients lines of all filters (thus, $36=4 \times 9$, complex coefficients), this file contains 9 lines - because of the second dimension. For the 3D, the file contains 9 lines and all of them contain 486 ($6 \times 9 \times 9$) elements.

Thoses coefficients are used in the `GenerateCoefficients()` function in the `itkBuildingMorphonFilters` library to create filters. The filters are implemented as `QuadFilter` (library `itkQuadFilter`) objects inherited from `itkComplexNeighborhoodOperator` library. This is exactly the same as the classical `NeighborhoodOperator` library except that we changed the line 144 of the `itkNeighborhoodOperator.h` file from

```
typedef std::vector<double> CoefficientVector;
```

to

```
typedef std::complex<double> ComplexType;
typedef std::vector<ComplexType> CoefficientVector;
```

This was done to support convolution with complex numbers and ensure better efficiency. To the best of our knowledge, it was the best way to ensure complex convolution with a minimum of changes.

The main useful function (except constructor) may be the function used to access to a filter, declared as:

```
ComplexOperator* GetComplexOp( int ID )
```

in the `itkBuildingMorphonFilters` library, with `ID` as an id of the filter: an integer between 1 and 4 in 2D (and 6 in 3D) and `ComplexOperator` is defined as

```
typedef itk::QuadFilter<ComplexType, ImageDimension > ComplexOperator;
```

Computation of the deformation field

The deformation field is computed in the `itkComputingMorphonDeformationField` library in the

```
ComputeUpdate(const NeighborhoodType &it, void *gd, const FloatOffsetType &offset)
```

function. This function computes the deformation field locally. Notice that the deformations are expressed in real space (hence, multiplied by the pixel spacing in every dimension).

The smoothing

The Gaussian filtering is used to: smooth the images before resampling at each resolution step and smooth the deformation field and certainty matrix at each iteration step. As presented in [2], we implemented 1D Gaussian filters to have an efficient algorithm. The standard deviation of the Gaussian kernels used to smooth the images were computed to keep as much information as possible while reducing aliasing. The standard deviations of the Gaussian filters used to filter the deformation fields and certainty are given as input to the program (see section 3.1). The standard deviation is the same for all filters at each level.

The operations presented in Sec 2.3 are implemented in library `itkComputingMorphonDeformationField` in the `ApplyUpdate(TimeStepType dt)` function.

3.3 The 2D and the 3D cases

The algorithm works on 2D and 3D images. Unfortunately, the 3D is slow. According to us this is due to the convolution which doesn't have a really efficient implementation in ITK and, unfortunately, this algorithm uses 4 or 6 convolutions of the deformed prototype image at each iteration at each level -with non separable quadrature filters. In addition, it also requires 3 convolutions for the smoothing of the field, of the certainty matrix and of the squared certainty matrix. For the fixed image, the convolution with the quadrature filters are only needed for the first iteration at each level because the image remains unchanged throughout a resolution level. Thoses convolutions could explain the slowness of the algorithm specially in the three dimensional case.

4 Results

The algorithm was tested with 2D and 3D images (which was stored in a .jpg, .png, .tif, .mha or .mhd format). The first image tested was the image which was presented in [11]. As illustrated on Figure 2(c), the output of the Morphon algorithm on this image is similar to the fixed image. The internal Gaussian kernels used to smooth images, fields and matrices are not exactly the same than in the original code. That

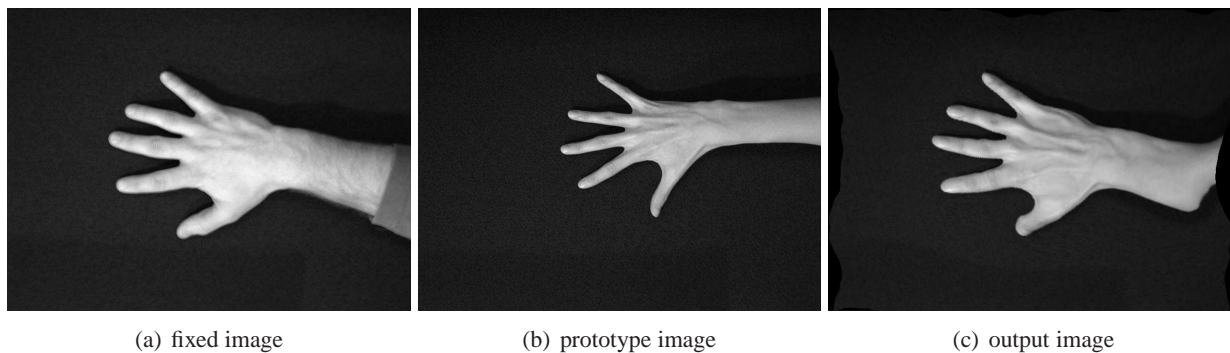


Figure 2: the hand figures: the fixed, the moving and the output images

explains the differences between our results and the results presented in [11].

To illustrate the result of the algorithm on 3D images, we registered two intra-patient lung CT images coming from a respiration correlated sequence (RC-CT). Figures 3(a) and 3(b) illustrate the inputs and the result. The images are presented with a vtk visualization platform: MedicalStudio ([9], <http://www.medicalstudio.org/>). The blue contours are the result of a simple threshold on the output image and illustrate the differences between the images.

5 Future work

An efficient convolution: an important and one of the most costly operation is the convolution. This operation takes approximately 70% of the computational time of an iteration. It's necessary to have a more efficient implementation to work on 3D images with reasonable time.

Support for other dimensions: For the moment, the method is limited to 2D and 3D images. This is due to the quadrature filters for which the coefficients are only computed for 2D and 3D images.

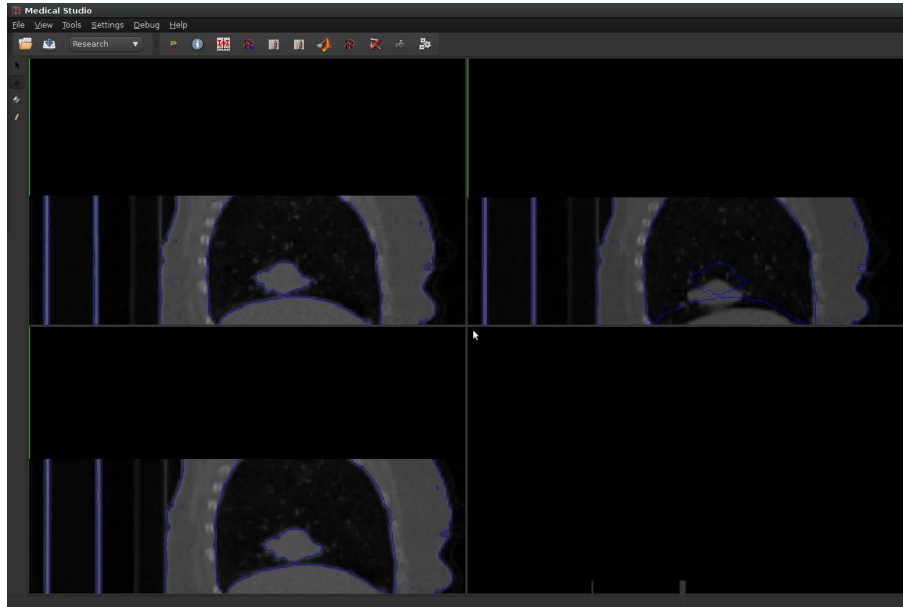
6 Conclusion

This method computes the registration between two images based on quadrature filter responses. It computes a certainty matrix and a deformation field.

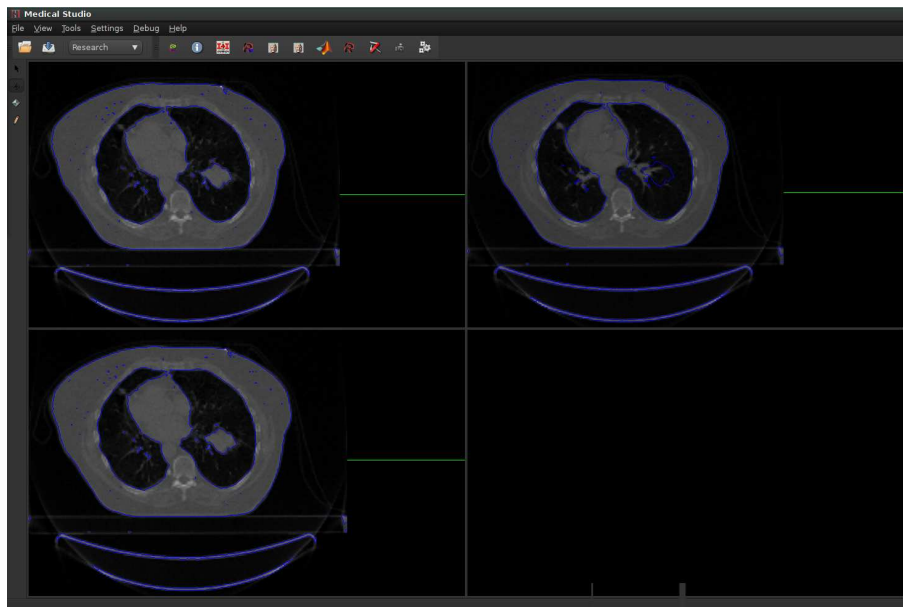
The implementation is constructed based on the ITK spirit. This implementation works for 2D and 3D images. To produce efficient algorithm we had to modify some classes. It was done in a way that only the necessary was modified.

7 Remark

During the calculation, the algorithm displays many information like the level, the current spacing, etc. But it also displays a number at the end of each iteration. This number is the sum of squared difference (SSD) between pixels of the image and not the actual metric used by the algorithm to compute the registration. Thus, you don't have to conclude that because this metric is growing up, the registration is going wrong.



(a) Sagittal orientation



(b) Axial orientation

Figure 3: The 3D inputs and output (sagittal and axial orientation): the fixed image (upper left), the moving image (upper right) and the Morphon's output (lower left)

8 Acknowledgements

Messrs Janssens, Orban de Xivry and Plumat thank the F.R.I.A. for its financial support.

9 Appendix

9.1 class diagram

References

- [1] S.K. Balci, P. Golland, and W.M. Wells. Non-rigid groupwise registration using b-spline deformation model. *insight-journal.org*, pages 1–8, June 2007. <http://hdl.handle.net/1926/568>. 1
- [2] Richard Beare and Gaetan Lehmann. Efficient implementation of kernel filtering. *insight-journal.org*, 6:1–18, July 2007. <http://hdl.handle.net/1926/555>. 3.2
- [3] Thirion J.-P. Image matching as a diffusion process: an analogy with maxwell’s demons. *Medical Image Analysis*, 2(3):243–260, Sept. 1998. 1
- [4] H. Knutsson. Representing local structure using tensors. In *The 6th Scandinavian Conference on Image Analysis*, pages 244–251, Oulu, Finland, June 1989. Report LiTH-isy-I-1019, Computer Vision Laboratory, Linköping University, Sweden, 1989. 2.1
- [5] H. Knutsson and C-F. Westin. Normalized convolution: A technique for filtering incomplete and uncertain data. Tromsø, Norway, May 1993. SCIA, NOBIM, Norwegian Society for Image Processing and Pattern Recognition. Report LiTH-isy-I-1528. 2.3
- [6] Hans Knutsson and Mats Andersson. Morphons: segmentation using elastic canvas and paint on priors. *Image Processing, 2005. ICIIP 2005. IEEE International Conference on*, 2:11–1226–9, Sept. 2005. 1, 2
- [7] Xavier Pennec, Pascal Cachier, and Nicholas Ayache. Understanding the demon’s algorithm: 3d non-rigid registration by gradient descent. *Medical Image Computing and Computer-Assisted Intervention*, pages 597–605, 1999. ISBN 978-3-540-66503-8. 1
- [8] Eduardo Suarez-Santana, Rafael Nebot, Carl-Fredrik Westin, and Juan Ruiz-Alzola. Fast block-matching registration with entropy-based similarity. *insight-journal.org*, pages 1–8, June 2007. <http://hdl.handle.net/1926/549>. 1
- [9] Daniela G. Trevisan, Vincent Nicolas, Benoit Macq, and Luciana P. Nedel. Medicalstudio: a medical component-based framework. *Workshop de Informatica Medica (WIM)*,, 2007. 4
- [10] Chia-Ling Tsai, C.V. Stewart, A. Perera, Ying-Lin Lee, Gehua Yang, and M. Sofka. A correspondence-based software toolkit for image registration. *Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on*, 5:3972–3977, Oct. 2006. 1
- [11] Andreas Wrangsj, Johanna Pettersson, and Hans Knutsson. Non-rigid registration using morphons. *Lecture notes in computer science (Lect. notes comput. sci.) ISSN 0302-9743*, June 2005. 2, 4