Image registration with unified particle swarm optimization

Dept. of Computer Science and Engineering, University of Ioannina

Author : Nikiforos Pittaras

October 2013

Acknowledgements

A sincere thank you to my supervisors and the people close to me for their continuous moral and material support.

Abstract

In this work we test the unified particle swarm optimization algorithm at the problem of image registration, using normalized mutual information, for rigid and non rigid applications. UPSO is tested for estimation of optimal parameters on rigid body registration reference problems and the parameter set result is applied on non rigid examples , using diffeomorphic grid - based warps. Our goal is to verify the efficiency of mutual information as a registration metric and the robustness of unified PSO.

Contents

List of Figures

List of Tables

Introduction

Image registration is the field of image analysis involved in efficiently estimating the correct mapping between correlated images. This is essential when processing or interpretation require the images to be aligned, a constraint not always met, i.e. when images are produced using different modalities. It has a wide range of applications, including medical imaging, computer vision and image reconstruction and has been the subject of a lot of research in the past years.

Image registration is usually set up with a fixed reference image and one or more images that are transformations of the first, called target or floating image(s). From here on we will assume a single floating image for simplicity. The goal of registration is to align the floating to the reference image via a transformation applied on the former, with the alignment quality measured by a similarity function. The transformation type is based on the modality and the application itself, ranging from simple rigid body and affine transforms to non rigid deformations. In this study we tackle rigid body transformation problems, and non rigid, grid-based diffeomorphic warps. After the transformation, interpolation is required to map the output to image space coordinates.

Since non trivial problems include a large or continuous transformation parameter space, an efficient optimization method is necessary to get results with acceptable accuracy and computation time. We will use the unified particle swarm optimization method (UPSO) for approximating a solution, a variant of PSO that allows additional control of the swarm's behaviour, directly affecting the algorithm's exploration, convergence and fine-grain search capabilities.

Selecting the similarity function depends on the registration strategy. Feature based registration includes a manual or automated selection of landmark points, assumed to contain enough image information that their alignment will correspond to the registration of the images themselves, as opposed to intensity based registration which takes into account all the pixels in the images to be aligned, with possible restrictions arising from the transformation - dependent overlap. Following the latter paradigm, we have selected normalized mutual information, a variant of mutual information showing robustness to variable overlap between the reference and floating images. Mutual information is a concept derived from information theory, extensively examined and widely used in image registration.

In figure 1.1 the general steps involved in image registration are illustrated. We will continue by analysing the main theoretical components of the process we follow and end by discussing the experiments with their results and our conclusions.

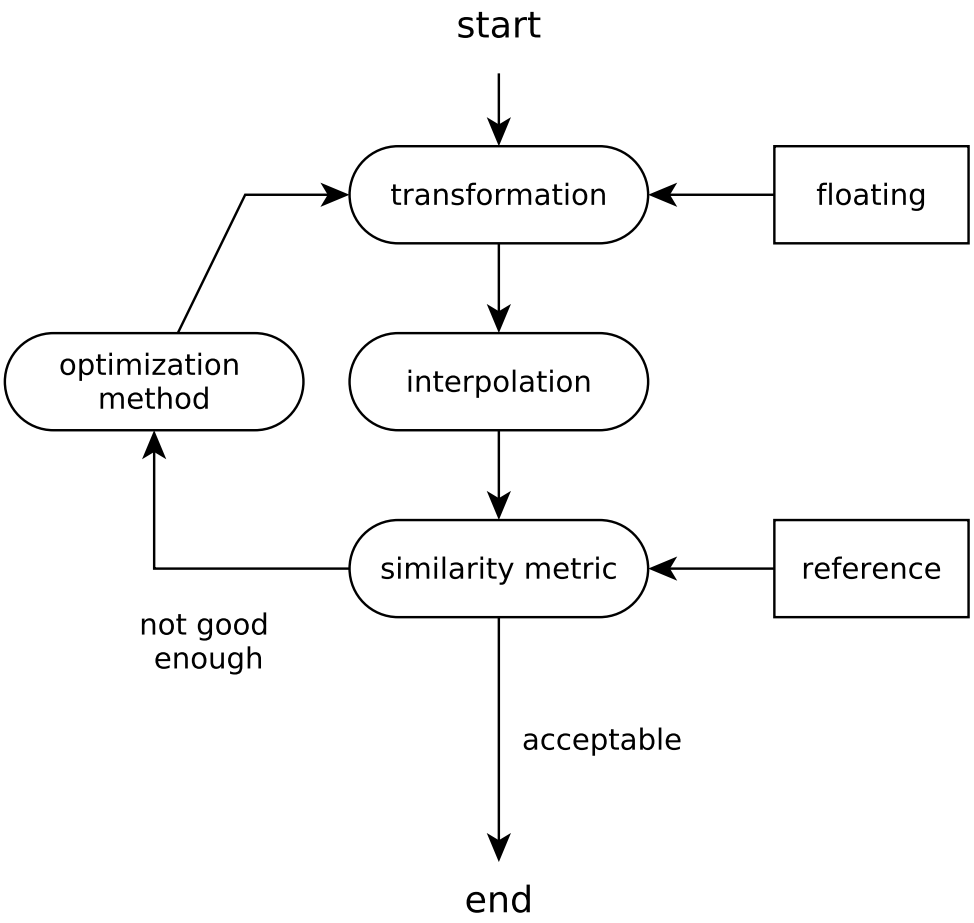

Figure 1.1: Image registration general steps.

1 Theoretical background

Overview

In this section we introduce the theoretical components of this study. We begin by describing the image transformation mechanisms for the rigid and non rigid case, followed by the similarity measure, mutual information. Lastly, we present PSO and its unified variant, the optimization algorithm used in our experiments.

1.1 Rigid body transformations

The first experimentation phase deals with rigid transformations. Recall the registration setup, where given two images (or image volumes), the reference and the floating, our aim is to find the transformation that best maps the latter to the former. In rigid registration, the floating image is assumed to be a product of a rigid body transformation, and such transformations will be used to match it to the reference. We will use two dimensional images, in grayscale format.

We continue by introducing rigid body transformations and its variants used.

1.1.1 Definition

A rigid body is an ideal description of an n-dimensional solid object, able only to be displaced and rotated, in $\mathbb{R}^n$.Let R be our reference image. Since R is considered a transformation of the floating image Fl, we can write the following.

$$R = T_{rigid}(p, Fl) \tag{1.1}$$

where $p = [\theta, t_r, t_c]$ is a parameter vector for the rigid transformation $T_{rigid}(\cdot, \cdot)$ that maps F to R, containing the rotation and row, column translation values respectively. Since we are dealing with two-dimensional images, we use an augmented, 2D transformation matrix:

$$M_{rigid} = \begin{bmatrix} cos\theta & -sin\theta & t_r \\ sin\theta & cos\theta & t_c \\ 0 & 0 & 1 \end{bmatrix} \tag{1.2}$$

This matrix rotates a 2D homogeneous coordinate vector θ radians around the rotation origin and then shifts it along the vector $[t_x, t_y]^\intercal$. Intuitively, a rigid transformation treats the image as a solid, non-deformable surface that can freely move and rotate but not curve or shear. Distances between points before and after the transformation are preserved, which means that angles, straight and parallel lines mantain their properties and values.

Figure 1.2: A scaled (left) and non scaled (right) counter-clockwise 30 degree rotation. The frame on the left is a scaled to fit the image, while the image on the right is cropped and the frame size is constant. We use the latter paradigm.

1.1.2 Forward and inverse mapping

The matrix 1.2 rotates around the origin $(0, 0)$, which in the context of image indices refers to an image corner, depending on the implementation (usually the upper left, either within or outside the image borders). To perform a rotation with respect to the image center of an $M \times N$ image, $N, M \in \mathbb{N}$, we use the following equation:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = M_{rigid}(p) \begin{bmatrix} i - mp_r \\ j - mp_c \\ 1 \end{bmatrix} + \begin{bmatrix} mp_r \\ mp_c \\ 1 \end{bmatrix} \tag{1.3}$$

where u, v are the transformed pixel coordinates, yielded by the transformation of the index positions i, j , $i \in [1 \dots M], j \in [1 \dots N]$.The point $[mp_r, mp_c]^\intercal$ is the midpoint of the image. This procedure moves the image so that its center coincides with the rotation axis, performs the rotation and translation and shifts back by $[mp_r, mp_c]^\intercal$. No scaling is applied and the resulting image retains the original size, meaning that points landing out of the image bounds are cut off and the image is cropped (figure 1.2).

The output pixel positions produced by 1.3 are used to assign the corresponding image intensities of the source pixel positions using the following:

$$I_t(u, v) = I_s(i, j) \tag{1.4}$$

where I_t, I_s the transformed and the source image respectively, u, v indices of the destination image and i, j indices of the source.

Equation 1.3, called a *forward mapping*, is a straightforward way of image mapping but has certain disadvantages. Since the resulting set of output pix-

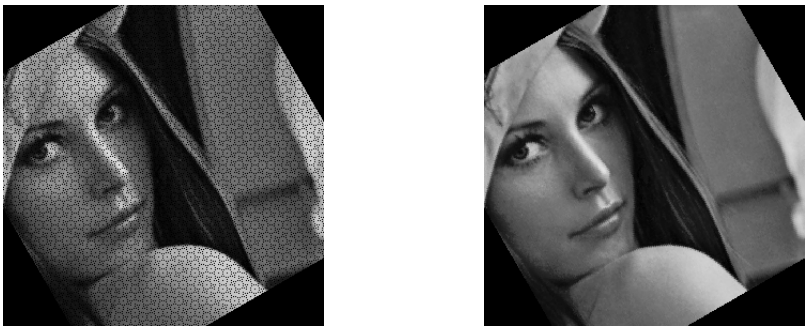

Figure 1.3: Examples of forward (left) and inverse (right) mapping of a 30 degree counter clockwise rotation. Since the image canvas is initialized to a default value (zero in this example) and forward mapping doesn't update some pixel positions, they show up as visible gaps in the output image.

els constitutes the transformation's range, if that transformation is not surjective, which is often the case, there will be gaps in the output image, namely pixel positions that did not get assigned to a source pixel. A way around this is to implement *inverse* or *backward mapping*:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = M_{rigid}(p) \begin{bmatrix} i - mp_r \\ j - mp_c \\ 1 \end{bmatrix} + \begin{bmatrix} mp_r \\ mp_c \\ 1 \end{bmatrix} \tag{1.5}$$

This time, i, j are the output image pixel coordinates, $i \in [1 \dots M]$, $j \in [1 \dots N]$, whose inverse transformation $M^{-1}_{rigid}(\cdot)$ produces a mapping to the source image position, u, v. Since i, j span the entirety of the output image, every output position will be assigned via the inverse transformation and no gaps occur. In short, a forward map connects each source pixel to an output position, whereas an inverse map assigns each output pixel to a source position. Figure 1.3 illustrates the difference between forward and inverse mapping. The intensity assignment is similar to forward mapping, with the indices switched:

$$I_t(i, j) = I_s(u, v) \tag{1.6}$$

1.1.3 Interpolation

Digital images are stored in the computer as arrays, with every cells' position and value representing the pixel's position and intensity respectively. This means that pixel indices have to be natural numbers (with the inclusion of 0, in most

programming languages). Since the transformation's output is generally a real number, the result has to be interpolated with respect to an integer cartesian grid. For u, v and i, j indices of the output and source images respectively, the intensity assignment takes the following form.

$$I_t(interp(u, v)) = I_s(interp(i, j)) \tag{1.7}$$

where I_t, I_s the transformed and the source image respectively and $interp(\cdot)$ an interpolation function. The specific application of interpolation functions is of course variable across programming languages, so 1.7 is just an outline of the process.

1.2 Diffeomorphisms

1.2.1 Overview

The second experimentation phase applies non rigid transformations to match the floating image to the reference. Such transformations are not limited to the capabilities of their rigid counterparts, but can apply virtually any effect on their input, including curving, shearing and arbitrary warps. Specifically, we apply diffeomorphic mappings, which are non rigid deformations (warps) with certain favourable properties.

1.2.2 Definition

Since our goal is to apply such transformations in image registration, they have to meet certain criteria, including *invertibility* and *smoothness*, so that given a point and its mapping, the existence of an inverse relation is guaranteed, and small changes in the transformation parameters yield small changes in the output. Transformations that satisfy the above are called *diffeomorphisms*.

The diffeomorphism construction procedure we follow is similar to grid based diffeomorphisms described by Cootes et al. [8], where a diffeomorphism is composed of several warps applied by the displacement of a rectangular grid and an affine transformation. We will continue by introducing the components of diffeomorphic mappings.

1.2.3 Affine transformation

An affine transformation is a transformation that preserves straight lines. Viewed as an expansion from the rigid transformation described in 1.1, it can additionally scale and shear its input, with the latter no longer considered as a solid object

but an elastic, deformable membrane. Having R,Fl as our reference and floating images, we can write:

$$R = T_{affine}(p, Fl) \tag{1.8}$$

where T_{affine} is the affine transformation function and p a parameter vector.

We use an augmented, 2D affine transformation matrix M_{affine}:

$$M_{affine} = \begin{bmatrix} a & b & t_r \\ c & d & t_c \\ 0 & 0 & 1 \end{bmatrix} \tag{1.9}$$

where t_r, t_c represent row and column translation, and the parameters a, b, c, d control rotation,scale and shear (see figure 1.4 for some examples). As with the rigid case, we keep the image frame fixed by discarding pixels outside its bounds, implement center-wise rotation by displacing the image by the negative of the center coordinates and use an inverse mapping technique to avoid uninitialized gaps in the output image :

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = M^{-1}_{affine}(p) \begin{bmatrix} i - mp_r \\ j - mp_c \\ 1 \end{bmatrix} + \begin{bmatrix} mp_r \\ mp_c \\ 1 \end{bmatrix} \tag{1.10}$$

where $[i, j]^\intercal$, $i \in [1 \dots M]$, $j \in [1 \dots N]$ the output pixel indices corresponding to the point $[u, v]^\intercal$ of the source image via the inverse transformation.

1.2.4 Grid - based deformations

Grid deformations use a set of control points in the image, arranged in a grid at integer locations. By displacing the grid nodes, we can shift every pixel in the image by a displacement vector, interpolated with respect to that pixel's position relative to the grid. Below a grid deformation function is illustrated.

$$I_t = w(p, Is) \tag{1.11}$$

were I_t, I_s are the transformed and source images, $w(\cdot, \cdot)$ the deformation function, and p the parameter vector containing the nodes' displacements.

We construct the grids recursively, with each level depending on the previous. For example, the first one consists of a single node at the image center. That node slices the image to four parts, and the second grid level has four nodes, each at the center of the slices. A level 3 grid has 16 nodes, and generally a level n grid contains 4^{n-1} nodes. Figures 1.5, 1.7 illustrate the grids' geometry and construction process.

The support of a node are all the pixels residing between that node and its immediate neighbours and includes the pixels that node can affect. If the grid

Figure 1.4: Reference image (upper left) and some examples of affine transformations.

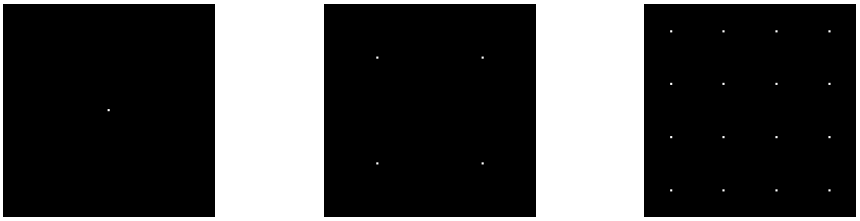

Figure 1.5: From left to right: deformation grid nodes of levels 1,2 and 3 respectively.

node i, j is displaced by a vector $d_{i,j}$, the displacement $d_{m,n}$ of every pixel in its support is given by:

$$\begin{aligned} d_{m,n} \;= & \; d_{i,j}\; k(|m - x_{i,j}|)\; k(|n - y_{i,j}|) + \\ & \; d_{i+1,j}\; k(|m - x_{i+1,j}|)\; k(|n - y_{i+1,j}|) + \\ & \; d_{i,j+1}\; k(|m - x_{i,j+1}|)\; k(|n - y_{i,j+1}|) + \\ & \; d_{i+1,j+1}\; k(|m - x_{i+1,j+1}|)\; k(|n - y_{i+1,j+1}|) \end{aligned} \tag{1.12}$$

where i, j are the node's indices in the *logical grid* space. This means that the nodes $(i, j), (i{+}1, j), (i, j{+}1), (i{+}1, j{+}1)$ form a rectangle, and the pixels within will be affected by these nodes alone. $x_{i,j}, y_{i,j}$ are the row and column coordinates of the (i, j)-th node in the image space, m, n are the pixel coordinates in the rectangle and $k(\cdot)$ is a kernel function to propagate the displacement of the nodes among the pixels. We use the following trigonometric kernel:

$$k(x) = \frac{1}{2}\,(1 + \cos(\pi x)) \tag{1.13}$$

where x is the normalized distance of the pixel from a node. When the four nodes defining a rectangle are equally displaced, the kernel moves that rectangle by that displacement without any distortions. Figure 1.6 illustrates the kernel. It can be shown that the transformation is guaranteed to be diffeomorphic, if all nodes are displaced by at most $|\frac{1}{\pi}|$ along each dimension. We can include virtual nodes with zero displacements outside the image borders for completeness and well defined node support regions.

1.2.5 Composition

We combine a number of grid deformations and an affine transformation to get the diffeomorphism F:

$$F_n(\cdot,\cdot) = T_{Affine}(w_1(p_1, w_2(\ldots w_{n-1}(p_{n-1}, w_n(p_n, \cdot))\ldots)) \tag{1.14}$$

$$I_t = F_n(p, I_s) \tag{1.15}$$

where T_{Affine} an affine transformation function, $w_n(\cdot,\cdot)$ a level n grid deformation function, $F_n(\cdot,\cdot)$ the diffeomorphism function, with a maximum deformation grid of level n and p the parameter vector containing the displacements of all grid deformations $w_i, i \in [1 \ldots n]$. I_t, I_s are the transformed and source image respectively. p_i is the subset of the parameters corresponding to the deformation

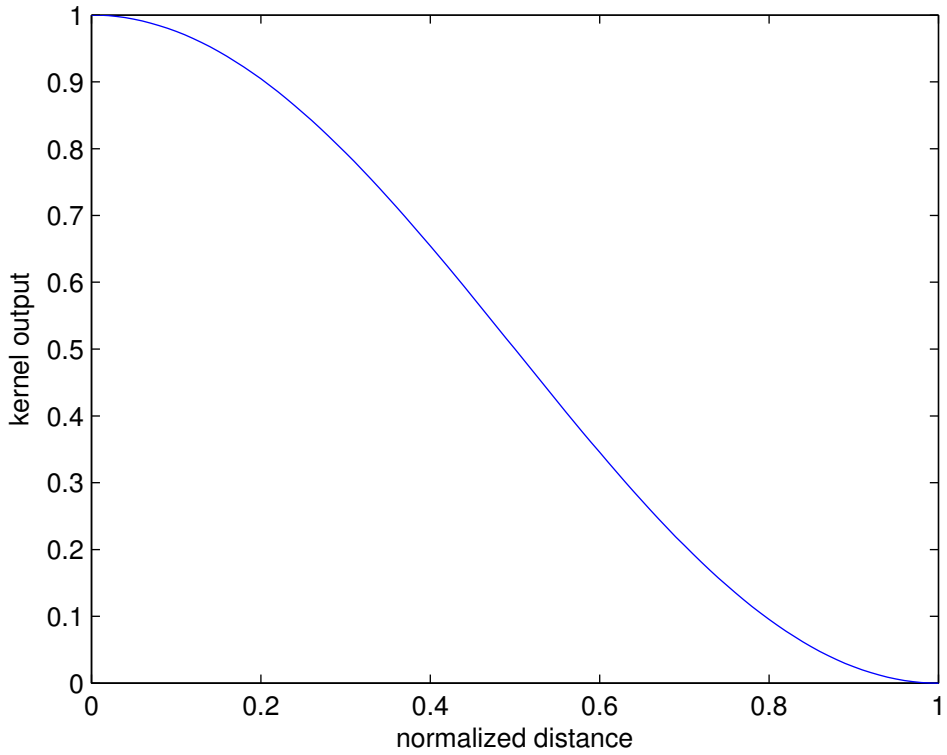

Figure 1.6: The cosine kernel.

w_i. Note that higher level (dense grid) deformations are applied first, followed by the lower level (sparse grid) deformations and the affine transformation. This means that low scale, fine grain deformations applied by the dense grids will be carried along with the larger scale warps since the former will be fully contained in the support of the latter. Figure 1.8 illustrates some examples of diffeomorphic transformations.

Figure 1.7: From left to right: level 1 grid, the slices it defines and level 2 grid.

Figure 1.8: Diffeomorphism examples of various parameters for maximum grid level from 1 to 4.

1.3 Mutual Information

1.3.1 Overview

Mutual information (MI) is a measure of shared information between random variables, one of the most studied similarity function in image registration, proposed as such by Viola and Wells [2] and Collignon et. al [1]. It is widely used due to its robustness and its few constraint requirements have made it popular in multimodal registration in medical imaging.

In order to be able to use MI to compare images, we model the latter as random variables, with pixel values representing event outcomes and pixel positions determining where and if these values coincide at an image pair. Since we are dealing with digital images, it is assumed that all random variables are discrete.

1.3.2 Definition

We begin by defining mutual information. Given two random variables X,Y, their mutual information I is given by

$$I(X,Y) = \sum_{x \in X} \sum_{y \in Y} p_{XY}(x,y) log_b \left(\frac{p_{XY}(x,y)}{p_X(x) p_Y(y)} \right) \tag{1.16}$$

where $p_X(\cdot)$, $p_Y(\cdot)$ are the marginal probability distributions of the random variables X and Y, and $p_{XY}(\cdot)$ their joint distribution. The log factor can be viewed as measuring a sum of distances between the current joint probability distribution $p_{XY}(\cdot)$, and the scenario where X and Y are independent, where the images have no shared information and their joint distribution is a product of their marginal distributions. Each distance is weighted by the probability of each value pair x, y. The log base b is hereby omitted and assumed to have the value 2, since it is convertible to any base, and information is usually measured in bits.

1.3.3 Intuition

To provide additional intuition on this similarity metric, we introduce the concept of Shannon entropy, $H(\cdot)$.

$$H(X) = -\sum_{x \in X} p(x) log(p(x)) \tag{1.17}$$

where $p(\cdot)$ is the probability distribution of X. Entropy can be viewed as a measure of uncertainty about a probabilistic outcome. Random variables with probability distributions approaching the uniform distribution will have entropy values approaching the maximum value of $log(n)$, where $\{x_1, x_2 \dots x_n\}$ is the set of

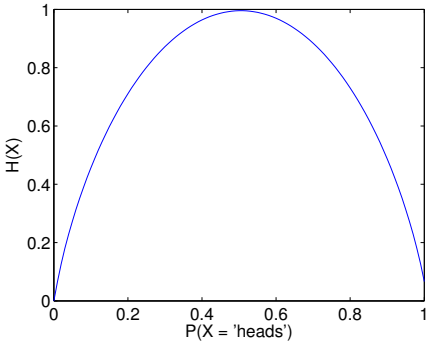

Figure 1.9: Entropy value versus the probability of $X = heads$.
Maximum entropy occurs at the event of a fair coin.

all outcomes. On the contrary, distributions favouring a subset of outcomes will have a lower entropy value and less uncertainty, since that particular subset of events will be more likely to occur. This is illustrated in figure 1.9 in an example of a coin toss. The definitions of conditional and joint entropy are given below.

$$H(X|Y) = -\sum_{x \in X} p_{X|Y}(x, y) log(p_{X|Y}) \tag{1.18}$$

$$H(X, Y) = -\sum_{x \in X} p_{XY}(x, y) log(p_{XY}(x, y)) \tag{1.19}$$

where $p_{X|Y}(\cdot)$ is the conditional probability distribution of X given Y. The joint entropy is the uncertainty surrounding the joint probability distribution of the random variables, while conditional entropy refers to the residual uncertainty of the one after the other is observed.

In order to have full knowledge about the result of a probabilistic experiment a priori, an amount of information equal to the amount of uncertainty must be available. In that sense, entropy and information are qualitatively similar measures, with the one representing the lack of the other. Combining 1.16 and 1.17, it can be shown that I can be expressed in terms of the marginal and joint entropies of the random variables.

$$I(X, Y) = H(X) + H(Y) - H(X, Y) \tag{1.20}$$

The following properties also hold.

$$I(X, Y) \geq 0 \tag{1.21}$$
$$I(X, Y) = H(X) - H(X|Y) \tag{1.22}$$
$$= H(Y) - H(Y|X) \tag{1.23}$$

Equation 1.20 shows that the mutual information surrounding two random variables X,Y is the sum of the marginal entropies (the case when X and Y are independent) minus the overall uncertainty associated with X and Y. For two unrelated random variables X and Y, it holds that $H(X, Y) = H(X) + H(Y)$ since no information is shared and equation 1.20 gives $I(X, Y) = 0$. Otherwise $H(X, Y) < H(X) + H(Y)$ and mutual information is the *uncertainty reduction* from the independent case to the configuration that yields the current joint entropy value.

Less residual uncertainty means a lower joint entropy value, thus more information is contained in the variable pair and the images represented by the random variables are better aligned. This is shown more clearly in property 1.22 (similarly,in 1.23), where the mutual information of X,Y, is the initial uncertainty about X, reduced by the residual uncertainty when Y is observed, $H(X|Y)$. In other words, MI is the information content Y has about X and vice versa.

In image registration, maximizing MI translates into seeking the transformation that yields image pairs X, Y with the most information content per image (large marginal entropies) and the most explanatory power they share about one another (small joint entropy).

1.3.4 Normalized variant

In intensity based registration all of the image pixels are considered in calculating a similarity value. This can be both redundant and detrimental to successful registration. Studholme et. al [4] have proposed a normalized variant of mutual information, where a transformation dependent subsection of the images is piped to the similarity metric calculation. Normalized mutual information is defined below.

$$N(X, Y) = \frac{H(X) + H(Y)}{H(X, Y)} \tag{1.24}$$

This MI variant has been shown to be more robust to cases with variable overlap. Here, the maximal NMI value of an image pair represented by X, Y will depend of their joint entropy with respect to the marginal entropies. This deals with the problem of inappropriately high MI values in suboptimal candidate transformations where the image background occupies a large portion of the transformed

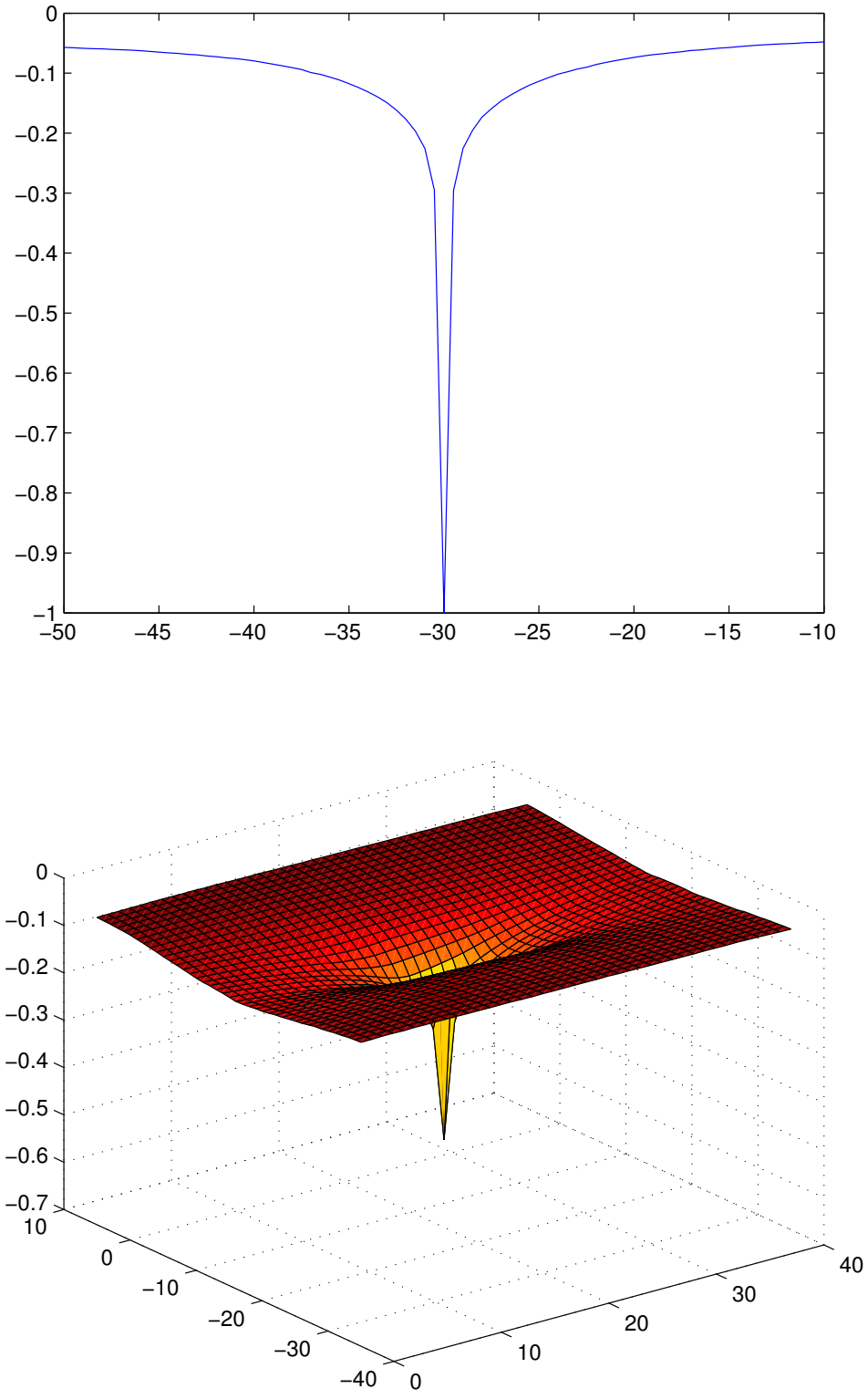

Figure 1.10: Minimum values for (negatively signed) MI for an image rotated by 30 degrees (top) and an image rotated by 30 degrees and translated row wise by -20 pixels(bottom). Note how the best objective value changes in accordance with the transformation's complexity.

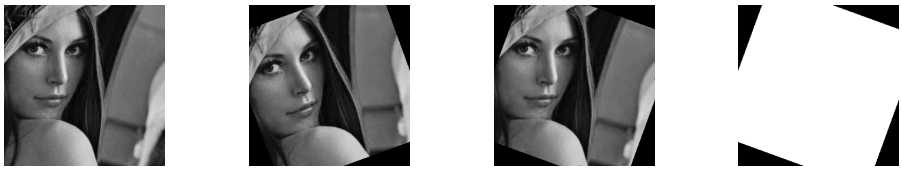

Figure 1.11: Reference (left), floating (center-left),transformed floating (center-right) and consideration overlap in white (right). In NMI, only the white part of the overlap is considered. Standard MI takes into account the black edges of the transformed floating image, match them with the corresponding positions of the reference, and lower the similarity value,despite the fact that the floating image is correctly transformed.

image, as well as backround - related biases of the standard function. We compute NMI in the overlap produced by each transformation. Figure 1.11 illustrates an example.

1.3.5 Properties

Mutual information works well as an objective function for registration, since its global optimum is significantly distinguishable from its local ones, especially in rigid transformations (see figure 1.10).

Section 1.1.3 describes the necessity of interpolating transformed pixels to the integer pixel grid. This operation introduces interpolation artefacts in the image that will in turn reduce the mutual information value of the floating and the reference images, even in cases of correct registration. This bounds the best possible similarity to a value lower than the self information of the reference image, and is an obstacle to setting an objective function threshold to the optimizer a priori. Instead, we test known transformations on images of the same modality and empirically set the MI threshold at a conservative estimation. In the worst case, the optimization algorithm will perform the maximum number of iterations. The introduction of such interpolation artefacts is more severe in cases of non rigid transformations. Figure 1.12 illustrates some interpolation methods on a warp deformation of a black and white square pattern.

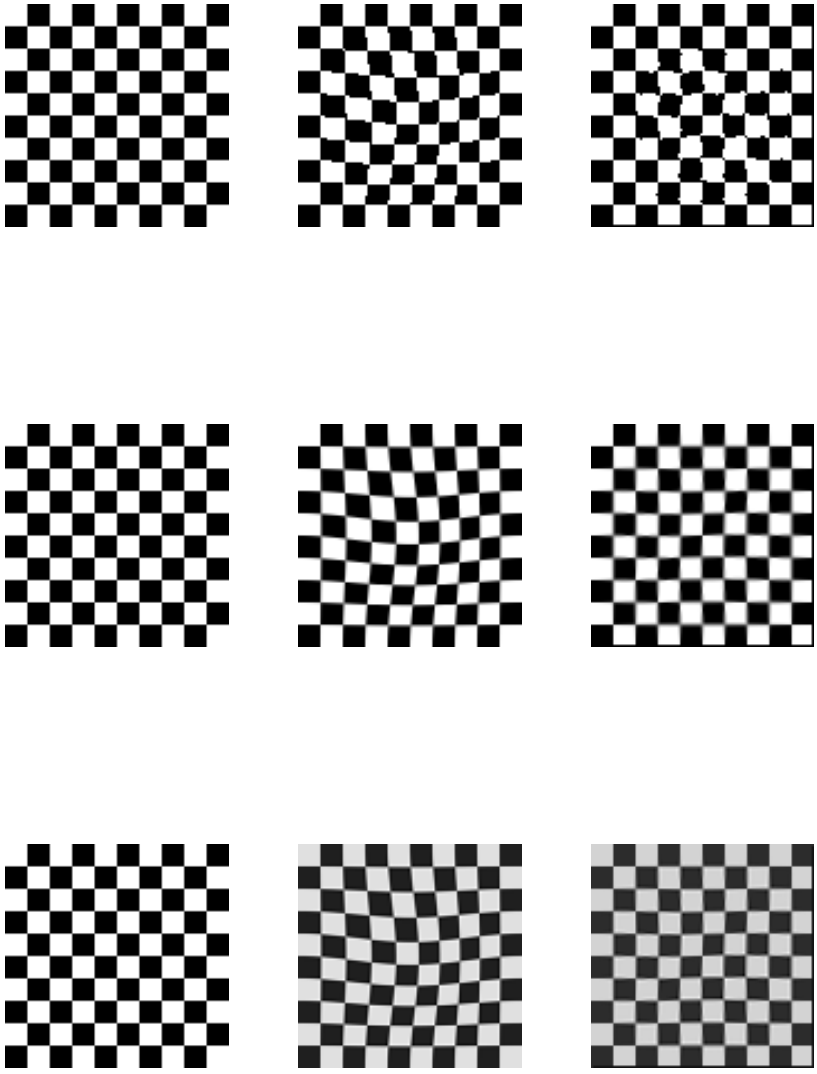

Figure 1.12: A level 1 warp with a displacement of 4 pixels along each dimension. From left to right : reference, initial floating, and estimated reconstructed image. From top to bottom: nearest neighbour, billinear and bicubic interpolation.

1.4 Particle Swarm Optimization

1.4.1 Overview

Particle Swarm Optimization (PSO) is a fairly recent member of the family of swarm intelligence algorithms, attributed to Kennedy, Ebenhart and Shi [5] [6]. A true representative of its class, it adopts a population based approach for finding an optimal solution, instead of mantaining a centralized control structure. As the name implies, PSO considers a group of candidate solutions, dubbed a particle swarm, that moves within the search space. At every step,each particle evaluates its current position via an objective function, and moves to a new position. The particle's displacement vector, dubbed velocity, depends on its own experience and the experience of the rest of the swarm, as private, local and global best previous positions influence each individual's new heading and speed.

PSO attempts to simulate natural structures of groups of living organisms, particularly flocks of birds and fish schools, where although precise and organized behaviour is observed, analysis and research indicate that such efficiency and complexity is the product of a handful of simple rules between individuals. This concept of cooperation towards a common goal has been established as a sustainable and advantageous strategy in biological evolution, given the right environment. PSO follows this paradigm through social sharing of information amongst the population. Unlike genetic algorithms, PSO lacks an explicit mechanism of combination and evolution of its population members, but focuses on the influence of social and individual behavioural factors in the swarm.

Because PSO is intrinsically distributed and decentralized, it can fully exploit modern parallel computing equipment, giving it an edge against traditional algorithms. It is computationally inexpensive, as it only requires the objective function's values in the search space, with no gradient information. It makes little to no assumptions about the nature of the problem, a fact that allows it to be used to a variety of applications under different conditions.

1.4.2 Algorithm and Parameters

We begin by declaring problem-specific parameters, namely the dimension d of the problem, the search space per dimension S and the objective function $f(\cdot)$. This allows the initialization of a swarm consisting of N individuals, where each particle x_i , $i = 1..N$ is a d-dimensional vector placed in the search space and bounded accordingly. Every one of these vectors represents a candidate solution to the problem and the quality of each can be measured by the objective function.

Every particle in the swarm has a memory that allows it to keep track of the best position visited, i.e. the particle's personal best. In addition, information about each particle's best position is communicated amongst the population, meaning

that every particle can acquire knowledge about the best position of its neighbours. That neighbourhood can be a subset of the swarm (the case of a local best of each particle , *lbest*) or the swarm itself (*gbest*), resulting in a classification of PSO variants as local and global PSO respectively.

Having said that, the next step is to move the particles within the search space, by calculating a velocity vector v for each one. Modern PSO variants use the following mechanism.

$$v_{ik}^{(t+1)} = \chi\left(v_{ik}^{(t)} + c_1 r_{1k}^{(t)}\left(p_{ik}^{(t)} - x_{ik}^{(t)}\right) + c_2 r_{2k}^{(t)}\left(p_{gk}^{(t)} - x_{ik}^{(t)}\right)\right) \tag{1.25}$$

$$v_{ik}^{(t+1)} = \chi\left(v_{ik}^{(t)} + c_1 r_{1k}^{(t)}\left(p_{ik}^{(t)} - x_{ik}^{(t)}\right) + c_2 r_{2k}^{(t)}\left(l_{ik}^{(t)} - x_{ik}^{(t)}\right)\right) \tag{1.26}$$

$$x_{ik}^{(t+1)} = x_{ik}^{(t)} + v_{ik}^{(t+1)} \tag{1.27}$$

where x_i, v_i, indicate position and velocity vectors respectively, $i \in [1..N]$ the particle index , $k \in [1..d]$ the dimension index and p_i is the personal best position of the i-th particle. Equation 1.25 is used in the global PSO scheme, where the index g is the index of the best particle in the swarm (where its personal best coincides with the global best). In local PSO (equation 1.26), l_i signifies the local best position in the neighbourhood of the i-th particle. In both schemes, the positions are updated using equation 1.27. The superscripts indicate the time step and r_1, r_2 are random number vectors, uniformly sampled from [0..1], used to guarantee a stochastic behaviour of the algorithm. The previous velocity value, $v_{ik}^{(t)}$,serves as an inertial bias that prevents drastic velocity changes of the particle, serving as a memory of the previous movement direction. The parameter χ, called the constriction coefficient, acts as a velocity suppressor to control the swarm's convergence to an optimal solution. The constants c_1 and c_2 are weights to each component of the velocity vector, appropriately called cognitive and social components respectively. Research on stability and convergence of the algorithm suggest the following relations of χ and c_1, c_2.

$$\chi = \frac{2}{2 - \phi - \sqrt{\phi^2 - 4\phi}} \quad , \quad \phi = c_1 + c_2 > 4 \tag{1.28}$$

Usual values of these parameters in related literature are $\chi = 0.729$ and $c_1 = c_2 = 2.05$, which are the ones we use.

Equations 1.25 and 1.26 are vector sums, scaled down by the parameter χ. The first term simulates an inertial behaviour, the second term is the contribution of the personal best position, while the third component is the contribution of either the single global best in the swarm, in the case of global PSO, or the local best of that particle, in the case of local PSO. These last two terms stochastically attract each particle in the swarm, and are called *cognitive* and *social* components of the velocity update respectively.

Following each particle movement, the swarm has to be evaluated again and the best positions recomputed. This goes on until an acceptable solution is found, or the maximum number of iterations has been reached.

The PSO algorithm can be summarized in the following steps.

PSO algorithm

1	Randomize swarm positions, set as best positions.
2	Evaluate swarm positions, find gbest/lbest value(s).
3	Calculate velocities and update positions.
4	Evaluate new positions.
5	Update personal and gbest/lbest positions.
6	If solution found or max iterations reached, exit.
7	Go to [3].

Table 1.1: PSO algorithm steps.

The algorithm exits, returning the globally best particle in the swarm.

1.4.3 Particle neighbourhood

While having a global best position in the swarm is unambiguous, in local PSO one has to specify the criteria by which a particle's neighbourhood is defined. This can be visualized by representing the swarm as a graph, with the vertices representing particles and the edges a neighbour relation between them. In this representation, global PSO's neighbourhoods are represented by a fully connected graph, as each particle is a neighbour of the other,and local PSO requires adopting a topology to describe the neighbourhood of each particle. Some examples are illustrated in figure 1.13.

Despite the intuitive way of geometrically defining each neighbourhood via a distance norm, this proved to cause the particles to form clusters, as the communication network of a group of neighbouring particles would be limited to a confined subset of the search space, where the local best would reside. A clustered population of particles would limit its search capabilities. Instead, the distance space used are the particle indices, with the $2m + 1$ adjoining particles $x_{i-m}, x_{i-m-1} \ldots x_{i-1}, x_i, x_{i+1} \ldots x_{i+m-1}, x_{i+m}$ making up the neighbourhood of particle x_i, including itself. We will call an m-sized neighbourhood one that a particle neighbours with m particles at either side in the ring (see figure 1.14). From here on, any mention of local PSO will assume a neighbourhood structured in the index space.

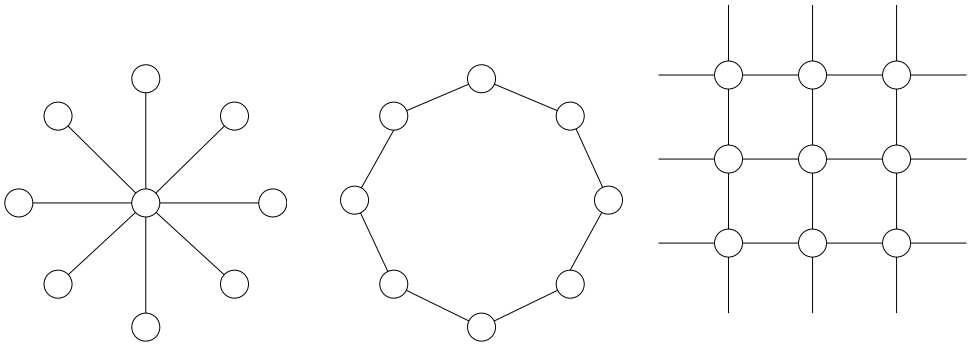

Figure 1.13: From left to right: Star,ring and von Neumann neighbourhood topologies.

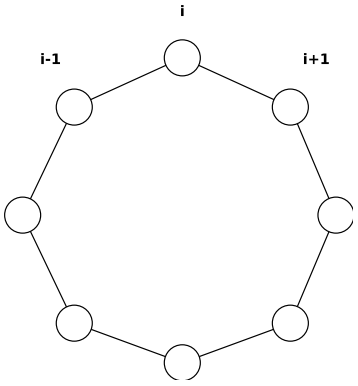

Figure 1.14: Index based ring topology with a neighbourhood of size 1.
The neighbours of particle i are the particles indexed $i \pm 1$ modulo N, where N is the number of particles.

1.4.4 Exploration and exploitation

As mentioned above, the neighbourhood size can classify PSO to its local and global variants. This classification is artificial, since global PSO can be viewed as a local PSO structure, with the local neighbourhood spanning the whole swarm. Nevertheless, the distinction is made due to the search efficiency the neighbourhood size allows and determines.

We say that global and local PSO exhibit different *exploitation* and *exploration* properties. In the first case, since the social component of every particle is the same, every individual in the swarm is attracted to that single position. This allows the algorithm converge faster (good exploitation), but also makes it prone to local optima as areas of the search space may not get visited at all (poor exploration). On the contrary, local PSO, lacking a universal attractor, allows the particles to perform a finer search of their surroundings, since the local best positions vary and are slowly propagated among the population via their index neighbours. This equips local PSO with better exploration capabilities, but limits its exploitation, because the local best attraction is varied and thus weaker. Of course, behavioural change can be achieved by altering the neighbourhood size and the social component coefficient, c_2.

Obviously, there is a trade off between exploration and exploitation, and selecting which PSO scheme to use depends on the application and the objective function used.

1.4.5 Unified PSO

Unified PSO (UPSO) is an attempt to utilize the strengths of each PSO scheme, namely the exploitation and exploration capabilities of global and local PSO respectively.

Recall from 1.25 and 1.26 the global and local velocity updates of a particle x_i. Acquiring these velocities:

$$G_{ik}^{(t+1)} = \chi\left(v_{ik}^{(t)} + c_1 r_1^{(t)}\left(p_{ik}^{(t)} - x_{ik}^{(t)}\right) + c_2 r_2^{(t)}\left(p_{gk}^{(t)} - x_{ik}^{(t)}\right)\right) \tag{1.29}$$

$$L_{ik}^{(t+1)} = \chi\left(v_{ik}^{(t)} + c_1 r_1^{(t)}\left(p_{ik}^{(t)} - x_{ik}^{(t)}\right) + c_2 r_2^{(t)}\left(l_{ik}^{(t)} - x_{ik}^{(t)}\right)\right) \tag{1.30}$$

we can calculate a composite velocity vector as the linear combination of the local and global velocities:

$$\mathcal{V}_{ik}^{(t+1)} = (1-u)G_{ik}^{(t+1)} + uL_{ik}^{(t+1)} \tag{1.31}$$

where $u \in [0 \dots 1]$ is a parameter called the *unification factor*, that controls the balance between the local and global velocity contribution. Setting $u = 1$, 1.31

results in the contemporary local PSO variant and setting $u = 0$ we get the global PSO variant. For $u \in (0 \ldots 1)$, both schemes contribute to the final velocity $\mathcal{V}_{ik}^{(t+1)}$, and we have an instance of unified PSO that combines the exploration and exploitation capabilities of both schemes. The position of the particles is updated using equation 1.27, using the composite velocity vector.

2 Experiments

Overview

In this section we present the experiments performed on rigid and non rigid problems. The simpler rigid case is tested first producing a favorable parameter set for the optimizer, on which we base the non rigid configuration later on.

2.1 Rigid case

The first experimentation phase tests the mutual information measure in rigid registration, using unified PSO as the optimization function. Our goals are to confirm mutual information's and UPSO's effectiveness as registration tools and discover an adequate parameter set for the optimizer, which we will use at the next phase, the more complex non rigid registration experiments.

2.1.1 Registration procedure

The problem is set up with a reference image R. We construct an initial floating image Fl with a set of parameters p, and plug the images in the UPSO algorithm to discover the corresponding parameter set of the inverse transformation.

Since the images are considered rigid bodies, the feasible solutions are the set of all 3-tuples $[\theta, t_r, t_c]^\intercal$, namely image rotation, row and column translation. To limit the search space, we have imposed maximum parameter values of $\pm\left[45, \frac{r}{3}, \frac{c}{3}\right]^\intercal$ to the parameter vectors, where r, c the image rows and columns number respectively. This limits rotation, row and column translation to absolute maxima of 45 degrees, and a third of the image height and width, a reasonable if not pessimistic assumption on registration problems. The initial floating image is constructed by a rigid transformation with randomized parameters in this valid parameter space. For a floating image produced with a parameter vector p, the transformation that correctly maps it back to the reference is parameterized with $-p$. This is achieved by using different trasformation functions:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) & t_x \\ sin(\theta) & cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} i - mp_r \\ j - mp_c \\ 1 \end{bmatrix} + \begin{bmatrix} mp_r \\ mp_c \\ 1 \end{bmatrix} \tag{1.32}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} i - mp_r + t_x \\ j - mp_c + ty \\ 1 \end{bmatrix} + \begin{bmatrix} mp_r \\ mp_c \\ 1 \end{bmatrix} \tag{1.33}$$

Equation 1.32 is the inverse mapping discussed in section 1.1.2 and equation 1.33 is the same transformation, with the difference that translation is performed

prior to rotation.

The quality of the candidate solutions is measured via the normalized mutual information metric, negatively signed to model the optimization procedure as a minimization problem. The image pair is plugged in the UPSO algorithm which returns a particle x_g, the global optimum discovered in the search space:

$$x_g = \underset{x}{argmin}\ \text{NMI}(R, T_{rigid}(x, Fl)) \tag{1.34}$$

where $T_{rigid}(\cdot, \cdot)$ is the rigid transformation function. The particle x_g corresponds to the parameter 3-tuple that minimizes the objective function , $NMI(\cdot)$.

Every parameter 3-tuple is tested by creating a candidate floating image, so interpolation artefacts are introduced, as mentioned in 1.1.3, that limit the optimum value of the objective function. Figure 1.15 illustrates a flowchart of the general registration procedure.

2.1.2 Testing

We use a 300×300, 8-bit grayscale image of the traditional image processing benchmark, Lena, in .tif format for our reference, the same one used in previous sections' figures for various illustrations.

UPSO parameter optimization

In order to discover an adequate parameter setting for UPSO, we construct three test images, the first two being heavily transformed to distinguish a parameter configuration that yields favourable results, and the third to test that configuration. The following table presents the transformation parameters, and figure 1.2 illustrates the images themselves.

Test image number	θ	t_r	t_c
1	-40	-45	37
2	33	-20	41
3	5	-2	3

Table 1.2: Test images transformation parameters

The UPSO parameters we are interested in optimizing, are the unification factor u, the swarm size SS, the maximum number of objective function evaluations $fevals$, and an additional parameter v_{div}, used to modify the maximum particle velocity per dimension.

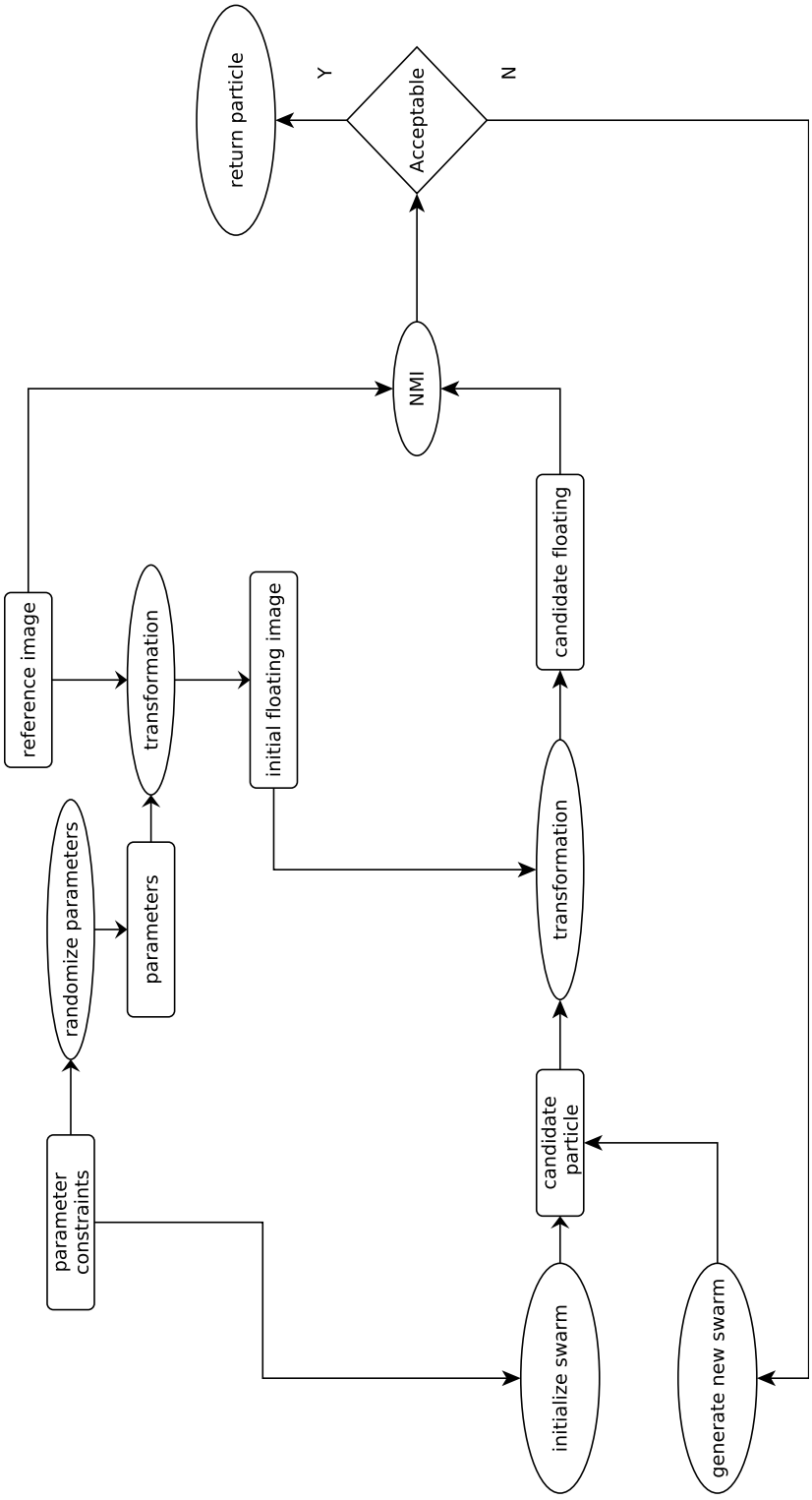

Figure 1.15: Registration flowchart.

Figure 1.16: From left to right: Test images 1,2 and 3.

For each parameter test image (test images 1 and 2) , we perform 100 experiments for possible values of u. Since $u \in [0 \dots 1]$, we divide that interval with a step of 0.1. We set the rest of the parameters vector of $[SS, fevals, v_{div}]$ to a default value of $[30, 3000, 3]$ and for every 20-tuple of experiments we change one of these to a candidate value.

Figures 1.18, 1.18 illustrate the success percentage and time elapsed per 100 experiments, versus the unification factor value. The generally poor scores are a result of the severity of the transformations of the test images 1 and 2. Images with milder transformations were registered too efficiently for a certain UPSO parameter setting to distinguish itself. Note that the first test image, being more heavily transformed (see table 1.2) performs worse than the other on both the time and success scores.

The plots show that a favourable setting for the unification factor resides in $[0.8, 0.9]$, as in that interval the success percentage peaks, and the total time elapsed drops. With regard to the rest of the parameters, the detailed experiment results seemed to favour a larger swarm size and a smaller particle speed, while larger function evaluations numbers did not have a significant impact. Setting the swarm size at a value of 60 particles, the velocity divider per dimension at the value of 13 and selecting 0.9 as the value for the unification factor, we performed 100 evaluation experiments on the third test image, using the following formula as an additional posterior check of the registration error.

$$E = \sqrt{\sum_{i=1}^{4} \left(c_{reg}(i) - c_{ref}(i) \right)^2} \tag{1.35}$$

where c_{reg} are the positions of the registered transformed image corners, and c_{ref} the position of the reference corners. This equation sums up the euclidean distances of the transformed image corners from the corresponding corners of the reference image. Obviously a greater distance means poorer registration, thus the

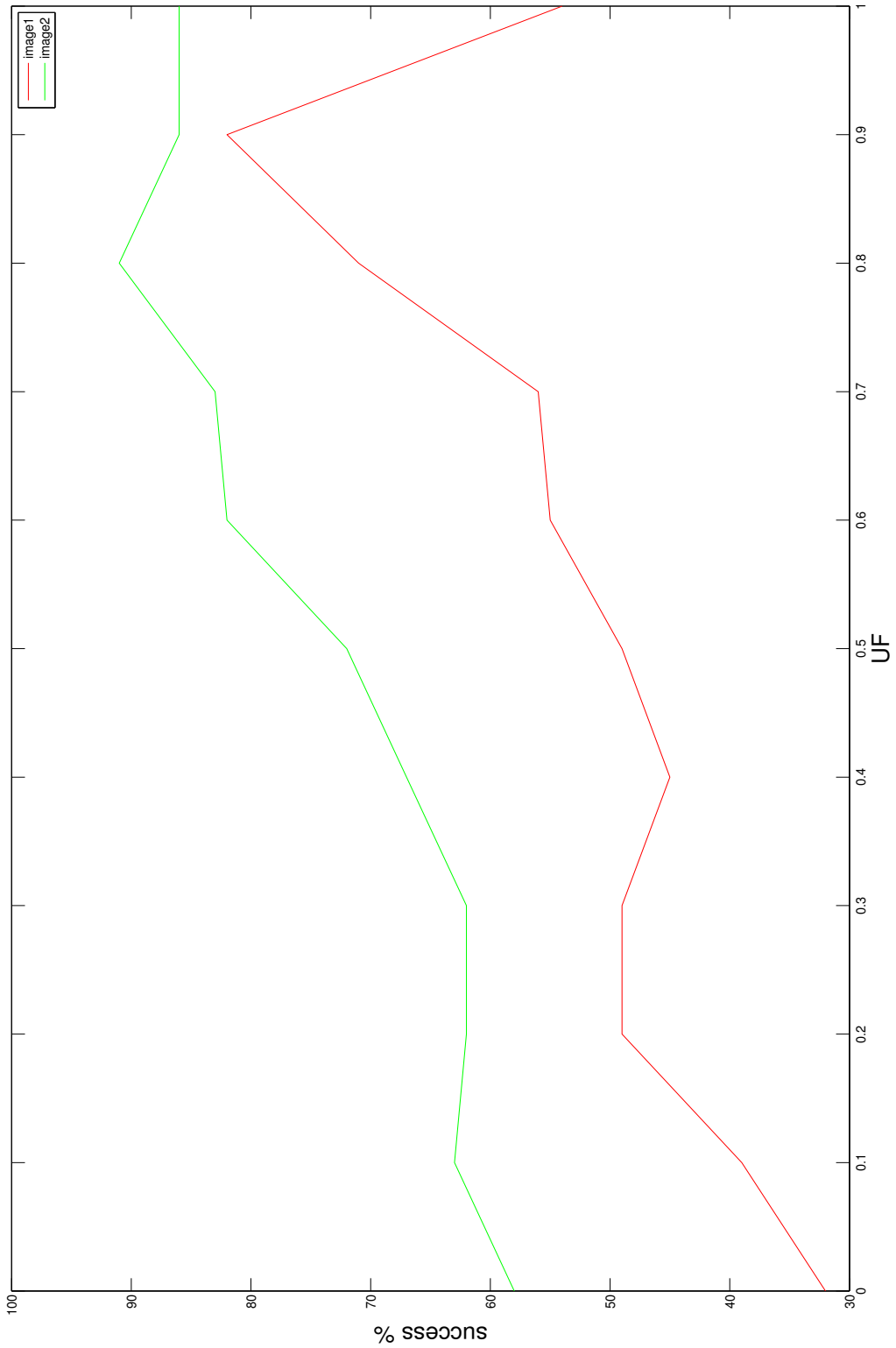

Figure 1.17: Success % versus unification factor value. An optimal value is at $[0.8, 0.9]$.

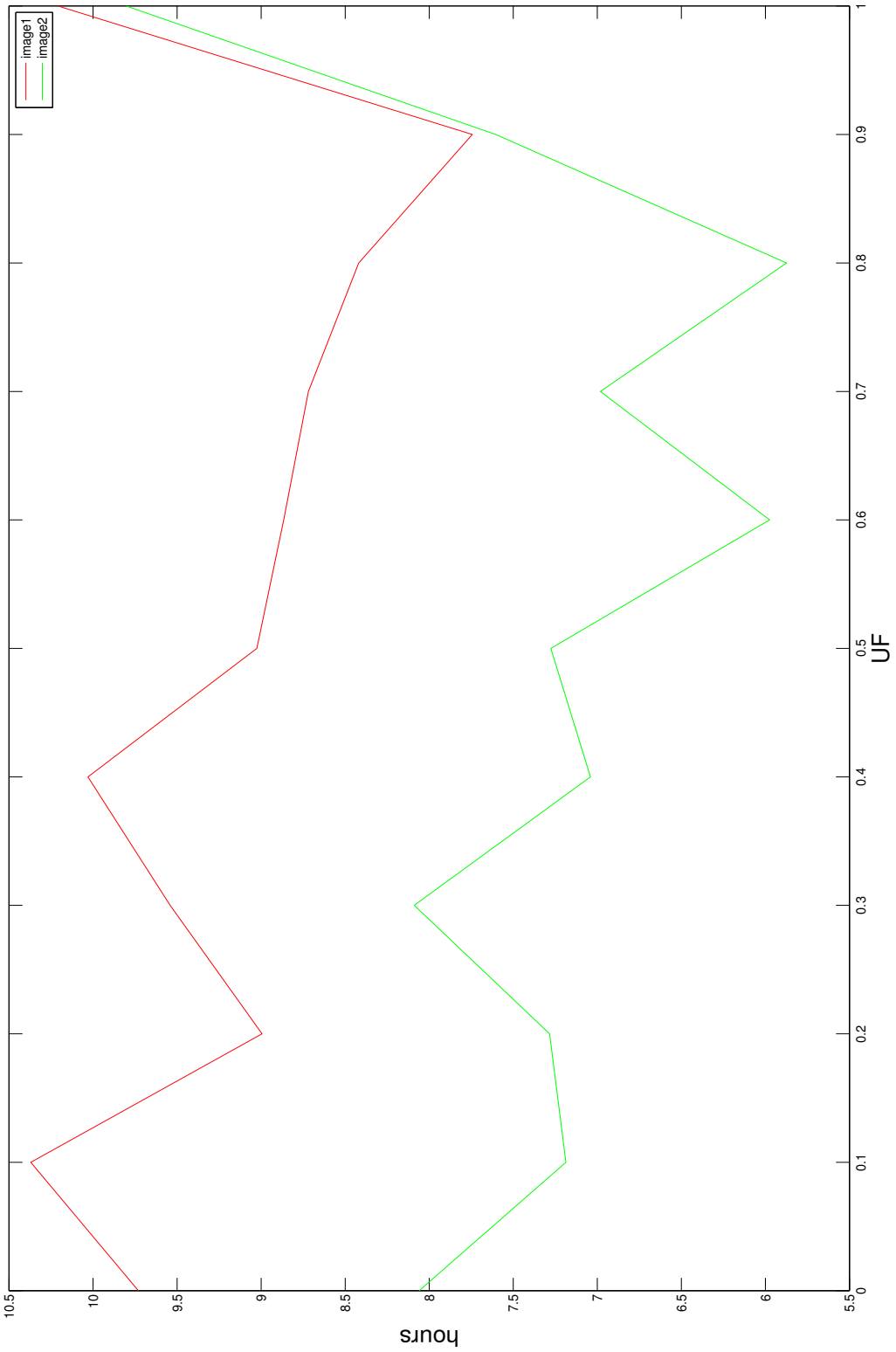

Figure 1.18: Time elapsed versus unification factor value. An optimal value is at $[0.8, 0.9]$.

error measure. Figure 1.19 displays the error of the evaluation experiments. There we can see that the set of PSO parameters $[u, SS, v_{div}, fevals] = [0.9, 60, 13, 3000]$ yielded very good registration results, achieving successful registration with sub-pixel accuraccy in all experiments, with the majority of error being distributed close to zero. Additional evaluation tests were performed on different evaluation images, with similar results.

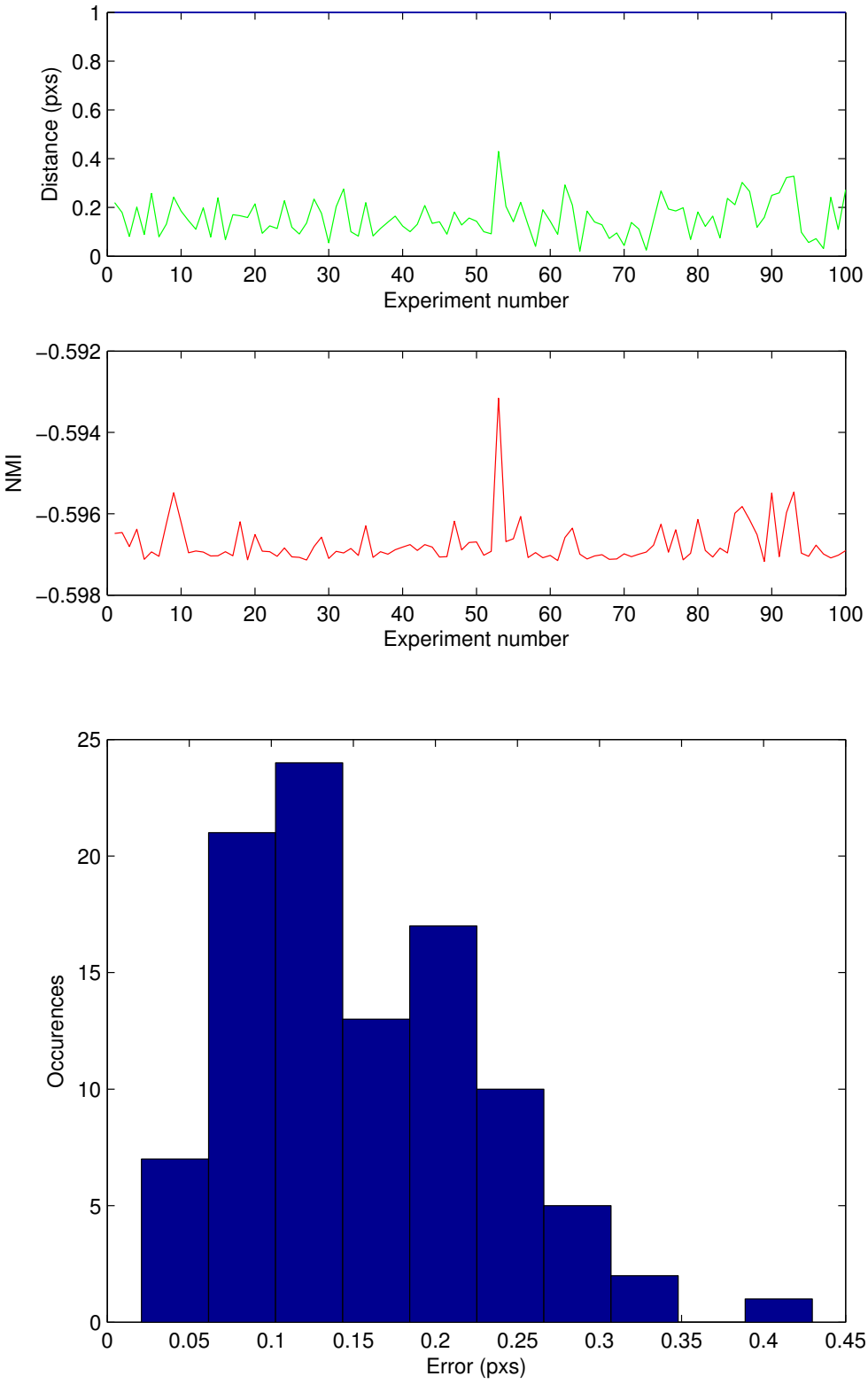

Figure 1.19: From top to bottom: Error distance in pixels, NMI score of each experiment and the distribution of error, for the evaluation image.

2.2 Non rigid case

In this set of experiments, we perform non rigid registration of images transformed with diffeomorphic warps. The parameter set used for UPSO is based on the one obtained in the rigid tests, with some changes depending on the problem size, which can get much larger than the rigid case. Additional changes and configurations are tested to try to improve the results of the standard parameter set, with varying degrees of success.

2.2.1 Registration procedure

We use the same reference image as with the rigid case. The affine transformation matrix parameters are randomized, distorting an identity matrix by quantities picked from $N\left(0, \frac{2max_a}{3}\right)$, where $max_a = 0.05$ a maximum distortion. The node displacement vectors are appropriately bounded to ensure the diffeomorphic nature of the transformation (according to 1.13), setting a maximum row and column displacement at $max_w = \left|\frac{1}{\pi}\right|$ and picked from $N\left(0, \frac{2max_w}{3}\right)$. The initial floating image F_{init} is constructed using equations 1.14 , 1.15.

A candidate floating image Fl_{cand} is computed in the reverse order of 1.14 to preserve the distortion of each warp:

$$Fl_{cand} = w_n(w_{n-1}(\dots w_2(w_1(T_{affine}(Fl_{init}))\dots)) \tag{1.36}$$

where w_n is a level n warp. The parameter vector is assumed to be passed accordingly according to 1.14 and is omitted. While Cootes et. al [8] decomposes the image step by step, dealing with each component at a time and constructing a new floating image at the end of each registration stage, we take a brute force approach and search the parameter space of all transformations simultaneously. This approach removes the dependency of each optimization stage on the previous one and the problem of early stage misregistration condemning the subsequent stages, especially in cases when image distortion elements are trying to be accounted for by the wrong grid levels. In addition, this process doesn't require intermediate floating image constructions, so interpolation artefacts are introduced only once. Thus, in the registration flowchart (figure 1.15), the transformation node includes all the components of the diffeomorphism, in the correct order described in the equation above.

A candidate solution vector is a concatenation of the 6 affine parameters with the nodes' displacements, resulting in a total of $6 + \sum_{i=1}^{n} 2^{2i-1}$ degrees of freedom, where n is the maximum warp grid level. The best particle x_g produced by the optimizer is defined below.

$$x_g = \underset{x}{argmin}\ \text{NMI}(R, F_n(x, Fl_{init})) \tag{1.37}$$

where $NMI(\cdot,\cdot)$ is the normalized mutual information function, $F_n(\cdot,\cdot)$ the diffeomorphism function, R the reference and Fl_{init} the initial floating image.

An estimation of the correct solution vector is obtained by calculating the parameters of the inverse affine transformation matrix and alternating the sign of the nodes' displacements. This is an estimation because interpolation noise skews the optimal solution away from the prediction, as illustrated in figure 1.12 and section 1.3.5. In addition, because the grid nodes are fixed, a displaced region in the initial floating image will have different distances during the construction and decomposition processes so the displacement vector magnitude will not be the same. A possible fix to this problem is proposed further below.

We start the experiments by performing 10-tuples of experiments, with PSO parameters at $UF = 0.9, v_{div} = 13, SS = 120, fevals = 100 \times SS$. Figures 1.20 to 1.22 illustrate the registration results, with the leftmost graph representing NMI score of each experiment, compared with the NMI score of the solution estimate (red line). The rightmost graph shows the euclidean distance of each solution to the estimated one. The experiments yield good results for affine only transformations and for diffeomorphisms with a grid level up to 2 (16 degrees of freedom), exceeding the NMI value of the estimated solution. The higher grid level experiments produced suboptimal results.

2.2.2 Improvement attempts

To cope with the size of the problem at grid levels 3 and 4, further UPSO parameter tweaking was attempted, using various techniques, including some discussed by Parsopoulos and Vrahatis [9].

- The trivial and expensive case is to increase UPSO's capabilities by adopting larger SS and $fevals$ parameters, to account for the large problem size. By increasing the swarm size to 200 and the maximum objective function evaluations to $150 \times SS$, the improvement was noticeable but the solutions remained suboptimal.

- Since MI contains insignificant local optima as mentioned in 1.3.5, the best particle at initialization would have a chance of being near the global optimum. Because of this, instead of having a constant value for u we apply an increasing unification factor scheme, at a linear rate. An ascending u from 0 to 1 (global to local PSO) would make the particles approach the best position at the early stage of the experiment and gradually perform a finer search as u approaches 1. Instead, this approach yielded worse results than the constant u case, even in the grid levels 0 through 2. This means that in large problem sizes, the good behaviour of MI as an optimization

function deteriorates and that the large search space lowers the probability of a particle being initialized near the optimum.

- An opposite scenario is to set u to decrease linearly from 1 to 0 (from local to global PSO). In this setting, the particles initially explore the search space, giving the swarm an opportunity to produce a better best particle before converging towards that position. This approach produced partially better results than the standard, constant u configuration. The NMI value absolute difference and the euclidean distance from the estimated solution are compared in tables 1.3 and 1.4 respectively, for grid levels 3 and 4, in the worst and best experiment cases.

  - This partial improvement shows that the exploration stage is crucial to the optimization procedure and should be applied early on. Certain experiments on grid levels 1 and 2, where the standard UPSO configuration succeeded, produced suboptimal results, most likely because the exploration stage was temporally or quantitatively inadequate (small portion of the experiment spent on exploration, with insufficiently large u values). Other approaches could adopt a non linear increase, such as a predefined quantized set favouring large values for u early on for the better part of each experiment, or an exponential increasing unification factor.
  - The mismatch between the universal improvement of NMI score and the non universal improvement of the distance from the estimation could arise from the lack of a fixed test image for these experiments, as each case was tested with randomized floating images.

Additional possible improvements are listed below.

- The solution of a diffeomorphism to be tested could be estimated better by allowing the grid nodes of a diffeomorphism during reconstruction to move prior to displacement. If d_n is the displacement vector of node n during construction and d_p the distances of all pixels in its support, pixel-node distances during reconstruction would be adjusted by $-d_n$ in equation 1.12. This moves the node to the position it had during construction and produces the exact inverse deformation, since the pixel-node distance in that region remains the same.This approach however might introduce some complexity in defining each node's support during reconstruction.

- In specific applications, where the approximate deformation regions can be pinpointed prior to registration, a dimensionality reduction technique could discard some nodes or even whole grid levels.

Decreasing unification factor			Fixed unification factor (standard)		
Level	Best case	Worst case	Level	Best case	Worst case
3	0.16×10^{-2}	0.44×10^{-2}	3	0.45×10^{-2}	0.104×10^{-1}
4	0.115×10^{-1}	0.181×10^{-1}	4	0.31×10^{-1}	0.402×10^{-1}

Table 1.3: Comparison of fixed versus decreasing (local to global PSO) u, illustrating the NMI distance from the estimation at the worst and best case.

Decreasing unification factor			Fixed unification factor (standard)		
Level	Best case	Worst case	Level	Best case	Worst case
3	0.6284	0.8767	3	0.5666	0.9245
4	2.1128	2.6210	4	2.3861	2.5368

Table 1.4: Comparison of fixed versus decreasing (local to global PSO) u, illustrating the euclidean distance from the estimated solution at the worst and best case.

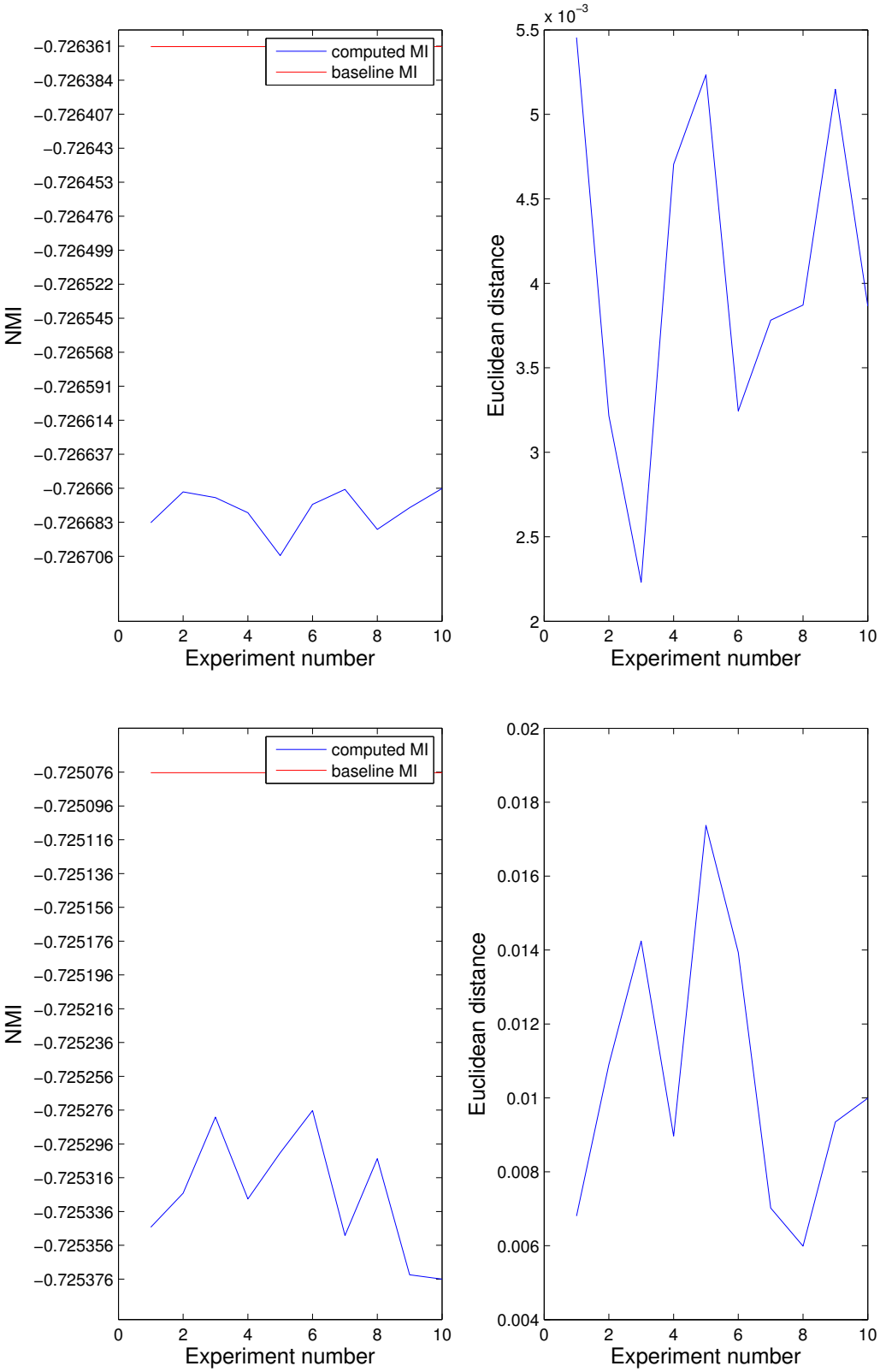

Figure 1.20: Affine (top) and warp level 1 (bottom) registration results.

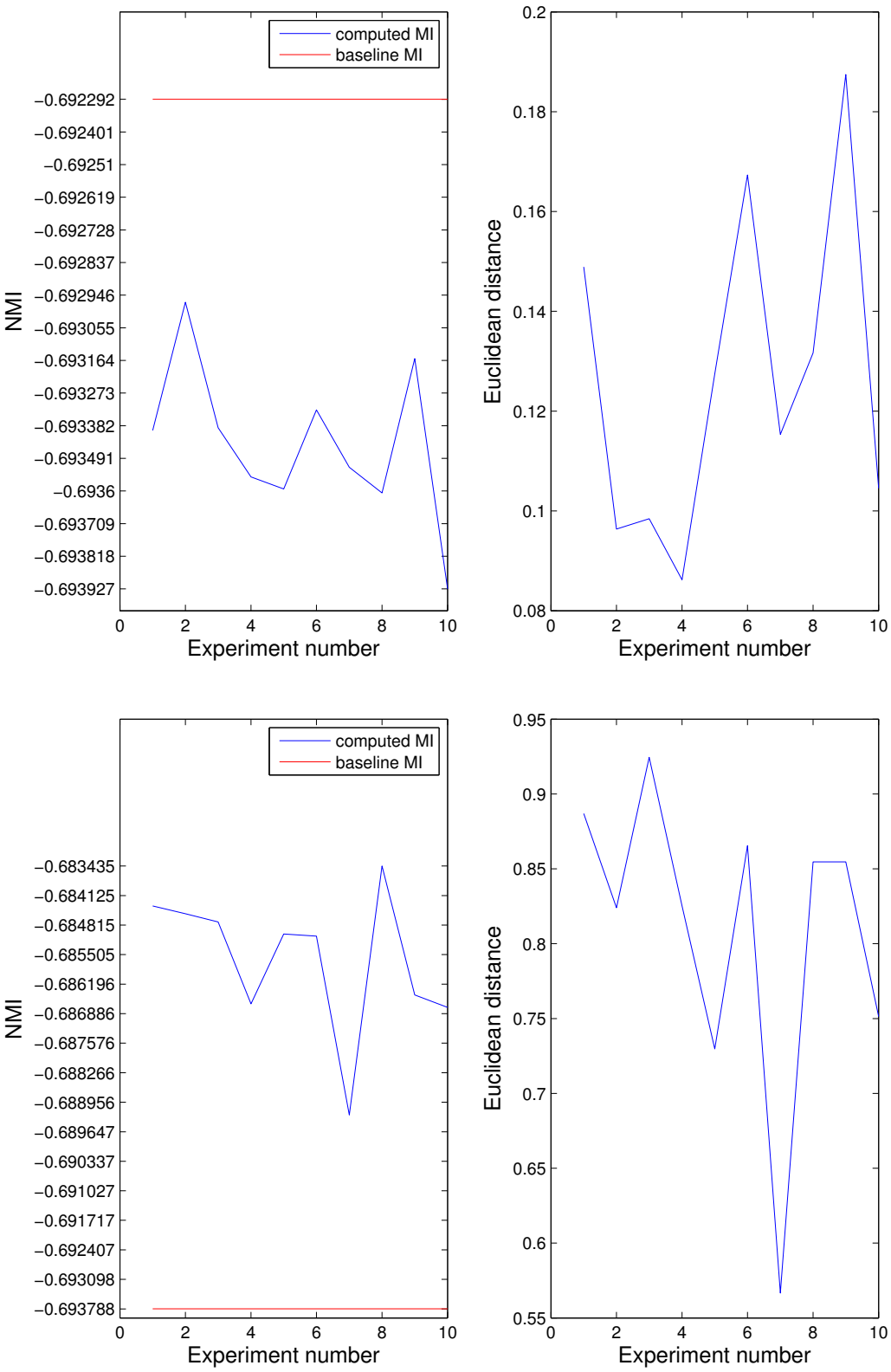

Figure 1.21: Warp level 2 (top) and 3 (bottom) registration results.

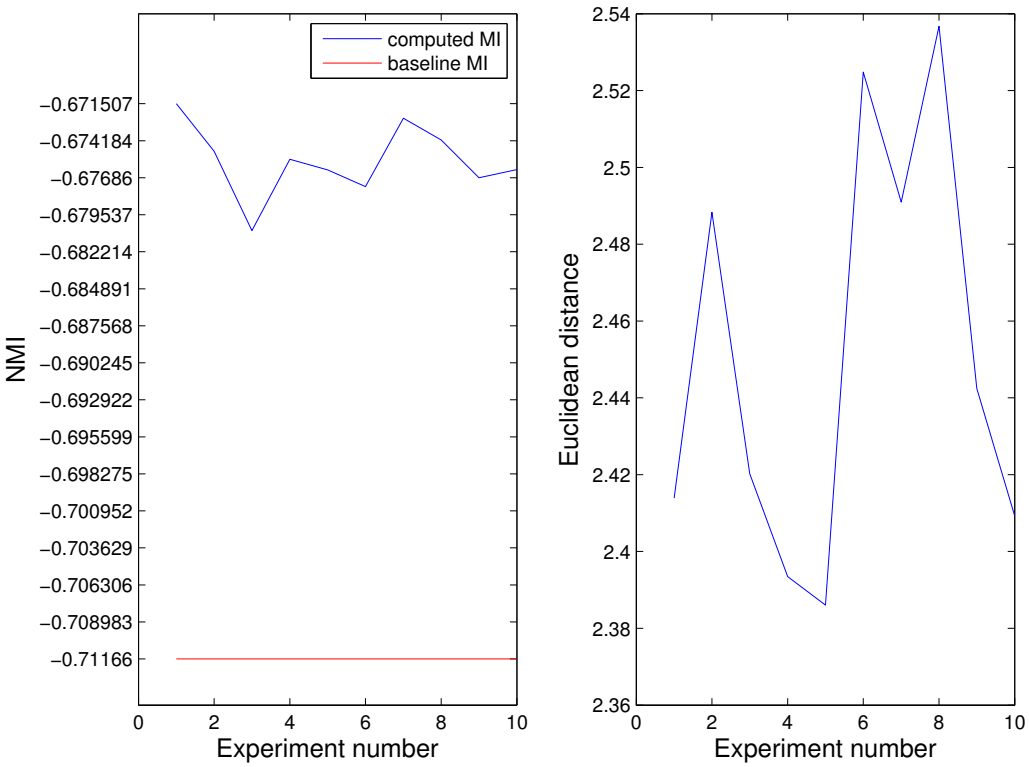

Figure 1.22: Warp level 4 registration results.

2.3 Conclusion

Registration with mutual information and UPSO has been shown to produce good results in the problem of rigid registration. In the non rigid case, the algorithm's efficiency reduces due to the large problem size associated with diffeomorphisms with a high deformation grid level.While some UPSO parameter settings have been explored in this study, the algorithm's plethora of parameters allows for further manipulation and optimization of its performance. Other UPSO configurations, adjustments in calculation and problem structure could provide better exploration as well as exploitation capabilities and serve as a robust, inexpensive tool for efficient image registration.

Bibliography

[1] A. Collignon, F. Maes, D. Delaere, D. Vandermeulen, P. Suetens, and G. Marchal. Automated multimodality medical image registration using information theory. Proc. 14th Int. Conf. Information Processing in Medical Imaging (IPMI95), vol. 3, Computational Imaging and Vision, Y. Bizais, C. Barillot, and R. Di Paola, Eds., Ile de Berder, France, June 1995, pp. 263274.

[2] P. Viola and W. M. Wells III. Alignment by maximization of mutual information. Proc. 5th Int. Conf. Computer Vision, Cambridge,MA, June 1995, pp. 1623.

[3] F. Maes, An. Collignon, D. Vandermeulen, G. Marchal and P. Suetens. Multimodality image registration by maximization of mutual information. IEEE transactions on medical imaging, vol. 16, no. 2, April 1997

[4] An overlap invariant entropy measure of 3D medical image alignment. C. Studholme , D.L.G. Hill , D.J. Hawkes Pattern Recognition 32, 1999, pp. 71-86.

[5] J. Kennedy, R.C. Eberhart. Particle Swarm Optimization. Proceedings of IEEE International Conference on Neural Networks IV, 1995, pp. 19421948

[6] Y. Shi, R.C. Eberhart. A modified particle swarm optimizer. Proceedings of IEEE International Conference on Evolutionary Computation, 1998, pp. 6973.

[7] K.E. Parsopoulos, M.N. Vrahatis. UPSO: A unified particle swarm optimization scheme. Lecture series on Computer and Computational Sciences, vol. 1, 2004, pp. 868-873.

[8] T.F. Cootes , C.J. Twining, K.O. Babalola and C.J. Taylor. Diffeomorphic statistical shape models. Image and Vision Computing 26 (2008), pp. 326-332.

[9] K.E. Parsopoulos, M.N. Vrahatis. Parameter selection and adaptation in Unified Particle Swarm Optimization. Mathematical and Computer Modelling 46, 2007, pp. 198213.