Non-linear Registration Using Block-Matching and B-Splines

Diplomarbeit

Heike Hufnagel

Ausgegeben von

Prof. Dr. rer. nat. Heinz Handels

Betreut von

Dr.-Inf. Xavier Pennec

Dr.-Inf. Gregoire Malandain

INRIA, Sophia Antipolis

2004

(Eingereicht May 25, 2004)

Statement of Originality

The work presented in this thesis is, to the best of my knowledge and belief, original, except as acknowledged in the text. The material has not been submitted, either in whole or in part, at this or any other university.

Ich versichere an Eides statt durch meine Unterschrift, dass ich die vorliegende Arbeit selbstaendig und ohne fremde Hilfe angefertigt und dass ich alle Stellen, die ich aus anderen Quellen entnommen habe, also solche gekennzeichnet habe.
Diese Arbeit hat in dieser oder aehnlicher Form noch keiner anderen Pruefungsbehoerde vorgelegen.

Luebeck, 31.05.2004

Abstract

Medical image registration techniques are of particular importance to improve and facilitate working with images of the body interior. The main idea in medical image registration is to find a transformation that maps a source image as close as possible to a target image by determining a corresponding position for each point of the source. To find a mapping between two images that takes anatomic variabilities into account, the respective transformation must be non-linear.

In this thesis, an algorithm is implemented that calculates a non-linear transformation to register 2D or 3D images. The transformation estimation is based on a block-matching technique that identifies a displacement field between two images. Using this set of correspondences as sampling points, the transformation is approximated by cubic B-Spline functions. In order to compute the approximation, a Least Squares estimator is employed which is extended in a second step to a weighted Least Squares approach to gain robustness. The minimization is solved by the conjugate gradient algorithm.

For a high accuracy of the result, the transformation estimation is performed in an iterative manner that is in addition nested in a pyramidal approach. For reasons of noise influence and lack of sampling points in homogeneous regions, a biharmonic regularization term is integrated to smooth the unfavourable effects.

Results are presented obtained by experiments on different artificial testing images as well as on images of histological slices of the brain. As highlighted, the non-linear approach yields satisfying registration results for most ...?.
The implementation of the B-Spline transformation completes a program that registrates two images fully automatically in a linear or non-linear way.

Acknowledgements

List of Figures

Contents

CHAPTER 1

Introduction

In 1895, the German physicist W.Roentgen discovered the first technique to see the inside of a three-dimensional structure without opening it when he found the phenomenon of the X-rays. Since then, a multitude of methods to gain visual access to the interior of a closed body has been developed. They are applied in a wide range of different areas to produce images; in this thesis, we deal with their application to medical images.

1.1 General Scope

Nowadays, medical images are widely used in health-care and biomedical research; the vast advantage to analyze the state and function of organs without actually having to perform a surgical operation is exploited in the most ways possible. Depending on the purpose, different methodologies to obtain the images are used. X-ray computed tomography (CT), for example, is able to represent tissue densities and atomic compositions whereas magnetic resonance imaging (MRI) is sensitive to proton density and relaxation times. Utilizing contrast agents (e.g. for an angiography) provides information on the permeability and function of tubular structures like blood vessels or organ ducts. The images of the body interior help the physician to diagnose a pathology, to examine the effect of a medication, or to evaluate the results of a surgery. Very often, he has to compare one image to another to analyse their feature differences and to draw a conclusion. A lot of image processing tools using computers have been developed to support the physicians doing these analyses.

The term *image registration* covers all procedures that are applied to align images with the intention to determine a relation between their corresponding features. The image registration process transforms one image (the *source*) to resemble another image (the *target*). Establishing the correspondences of spatial information between two images by an image registration is fundamental to medical image interpretation and analysis.

The two images to be registrated might be acquired with different sensors, in which case the operation is called *multimodal* registration. When registrating two images produced by the same kind of sensor at different times, the process is referred to as *monomodal* registration. There exist 2D-to-2D, 3D-to-3D and 2D-to-3D registration appliances, however, the registration of multiple 3D images such as MR and CT volumes is the most common concerning medical informatics approaches.

Medical image registration inter alia provides the means

- to compare two images in an analytical way, e.g. to survey a region of the body before and after a surgery,
- to fuse anatomical and functional information, e.g. to detect the accurate position of a tumor obtained by MRI in the skull (obtained by CT),
- to map an image to an atlas image, e.g. to do an automatic segmentation of the bones of the pelvis.

In this thesis, a method to registrate two- or three-dimensional monomodal images using an algorithm based on B-Splines is presented. Its implementation extends already existing program called `Baladin`. `Baladin` was created to align two-dimensional histological slices of the brain in order to form a three-dimensional brain model. It calculates linear (rigid and affine) transformations using a block-matching technique (see sections 2.1 and 2.2) [1].

The objective of this thesis is the development of a *non-linear* transformation class to upgrade the usage options of the `Baladin` program. The non-linear transformation estimation is realized by determining a B-Spline approximation of the desired transformation that is finally integrated in the `Baladin` program.

The content of the thesis is structured as follows:

In section 1.2, the issue of *image registration* as the domain where this thesis belongs is described elaborately.

In chapter 2 the knowledge and data where we start from are presented, that is, the *block-matching* algorithm, the program *Baladin* and the fusion of those three subjects forming the topic of this thesis are specified.

In chapter 3, the spline functions *B-splines* and their properties are defined and depicted.

In chapter 4 the application of B-spline functions to the transformation problem at hand is explained with all customization required.

Chapter 5 deals with the details of how the wanted transformation is calculated on basis of the algorithms that are implemented in the *Baladin* program.

In chapter 6 specifications concerning the functions used as well as the implementation of the algorithms are given.

Following, chapter 7 presents and discusses the results. Concluding, chapter 8 summarizes the outcomes and gives an outlook on the measures to be taken next.

The subsequent part of chapter 1 is subdivided in the following sections:

In section 1.2 an overview on the subject 'Image Registration' as well as the mathematical notation of image representation is given. In section 2.1 the block-matching algorithm is introduced and explained in detail. In section 2.2 the program `Baladin` and its features are described whereas section 2.3 depicts the idea and advantages of a non-linear registration.

1.2 Image Registration

Let I_S be the source image whereas I_T denotes the target image of the registration problem. In medical imaging, we deal either with 2D or 3D images. Next, we define the continuous image volume functions:

Definition 1.1 *Continuous Image Functions in 3D:*

$$\begin{aligned} & I_S : \mathbb{R}^3 \to \mathbb{R}, \quad \vec{x} \mapsto I_S(\vec{x}) \\ \textit{and} \quad & I_T : \mathbb{R}^3 \to \mathbb{R}, \quad \vec{y} \mapsto I_T(\vec{y}) \end{aligned} \tag{1.1}$$

In reality, images obtained by the sensors described above are never continuous functions but sampled on a regular grid, hence, they are defined on a discrete *pixel* or *voxel* basis. Here, “pixel” is the abbreviation for “picture element”, and “voxel” stands for “volume element”. The pixels divide a 2D space into a collection of uniform rectangles whereas the voxels divide a 3D space into a collection of uniform cuboids. Hence, a medical image I is represented by its collection of voxel (or pixel) values. Each value $I(\vec{x})$ specifies the grey scale intensity of the respective voxel at position $\vec{x}$.
In practice, we have to pay attention to the fact that the voxels are defined over a three-dimensional grid, that is over Ω_S and Ω_T respectively where the voxel sizes are usually neither standardized nor cubic. It holds:

$$\begin{aligned} & I_S : \Omega_S \to \mathbb{R}, \quad \vec{x} \in \Omega_S, \quad \vec{x} \mapsto I_S(\vec{x}) \qquad (1.2) \\ \text{and} \quad & I_T : \Omega_T \to \mathbb{R}, \quad \vec{y} \,\epsilon\, \Omega_T, \quad \vec{y} \mapsto I_T(\vec{y}). \end{aligned}$$

It is important to distinguish the *voxel* coordinates from the *world* (or *real*) coordinates when working with the image volumes.
Registrating two images I_S and I_T means searching for each voxel of I_S a corresponding position in I_T. To describe all correspondences, we introduce a *transformation* or *mapping* function T that assigns a new position $\vec{x}_{new}$ to each position $\vec{x}$ of image I_S:

$$T : \mathbb{R}^3 \to \mathbb{R}^3, \qquad T(\vec{x}) = \vec{x}_{new}.$$

T describes the relationship between two images regarding their features.
To determine the transformation T different approaches that depend on the characteristics of the problem at hand have been developed, three of them are introduced in the following:
The *landmark-based* registration is based upon identifying corresponding point landmarks in the two images. Using the information about their positions in the source and in the target image the algorithm calculates a transformation T that minimizes the differences between the deformed source ensemble and the target ensemble of points.
For the *surface-based* registration the surface features are segmented in the two images before the transformation T is computed by minimizing a certain kind of distance measure between them.

Different from the two registration algorithms described above, the *voxel-intensity-based* registration method works directly on the intensities of the two images; there are no requirements to segment corresponding structures. This type of registration uses a large portion of the image data, hence, it tends to average out any errors that are caused by noise [3, 4].

The simplest applications are the alignment of two images of the same modality showing the same subject or the alignment of two slices of the same image volume. The idea is to deform the source image in a way that the maximum of the similarity measure between the corresponding voxels of source and target image is found. The most intuitive way to determine the degree of similarity is to sum up all squared differences (sum of squared differences, SSD) between the intensities of the corresponding voxels, yet, this method works only for images of the same modality. In that case, alignment is adjusted until the smallest SSD is found [5].

Definition 1.2 *Sum of Squared Differences:*

$$SSD = \sum_{i=1}^{n} \| I_T(\vec{y}_i) - I_{S,new}(T(\vec{x}_i)) \|^2 \tag{1.3}$$

with I_T being the target image, $I_{S,new}$ being the deformed source image and $\vec{y}_i, \vec{x}_i$ the n voxels they are built out of.

For images of different modalities, the registration methods are often based on statistical relationships between voxel values (for more information see [21]).

There are various *transformation classes* used in the registration algorithms described above. They are defined by the space of research where the transformation T is calculated, therefore, the transformation classes have different degrees of freedom. Moreover, we distinguish between linear and non-linear registration methods. Actually, the term *linear* in this context is mathematically not correct as a linear transformation T between two vector spaces V and W with $T : V \to W$ fulfills

$$\begin{aligned} T(v_1 + v_2) &= T(v_1) + T(v_2), \quad v_1, v_2 \in V \qquad \text{and} \\ T(\alpha v) &= \alpha T(v), \quad \text{for any scalar } \alpha \end{aligned}$$

only if the translation term is neglected [7].

Below, the most common transformation classes are described [3], [6]:

- *Rigid* transformations are linear transformations that allow one rotation R and one translation t to align the two images. The standard three-dimensional representation looks like this:

  $$T(\vec{x}) = R\vec{x} + t, \qquad \vec{x} \in \mathbb{R}^3 \tag{1.4}$$

  with R being a matrix of dimension 3×3 and t being a vector of dimension 3×1. Matrix R contains three rotations of angles ϕ_x, ϕ_y, ϕ_z, $R = R_x R_y R_z$ with

  $$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\phi_x) & -sin(\phi_x) \\ 0 & sin(\phi_x) & cos(\phi_x) \end{pmatrix}$$

  determining the rotation around the x-axis and R_y and R_z the rotations around the y- and the z-axis respectively (see their appearance in appendix A). Thus, a rigid transformation in 3D contains 6 parameters of degrees of freedom. It is used e.g. when registrating two images of the same object merely in a different position or orientation.

- *Affine* transformations are linear transformations that preserve the parallelism, i.e. all points of the image lying on a line before the transformation will still lie on a line afterwards. Therefore, affine transformations allow to obtain torsion, shearing, compression and extension of features, its three-dimensional representation looks like this:

  $$T(\vec{x}) = A_{sc} A_{sh} R\vec{x} + t, \qquad \vec{x} \in \mathbb{R}^3, \tag{1.5}$$

  with A_{sc} being the matrix that contains the scaling parameters and A_{sh} being the matrix that contains the shearing parameters for each direction (details see appendix A). Affine transformations have 12 degrees of freedom and are used e.g. when registering an image with an atlas image or for the initialisation of all non-linear registration algorithms.

- *Non-linear* transformations must be applied when the two images in question possess some kind of features that cannot be aligned with an affine transformation because the features do not correspond with respect to the parameters of an affine transformation. This occurs e.g. when bones of different persons are to be registered to obtain patient-atlas mapping or when registrating intestines of the same patient taken at different times. In the first case, the anatomical variabilities have to be incorporated what is not possible with a linear transformation. The second case presents the problem of 'soft tissue deformations' that must be taken into account.

  When performing a linear transformation on an image, each voxel position is deformed with the same kind of operation, e.g. a multiplication with a certain matrix. In contrast to this, when applying a non-linear transformation to an image each voxel position is assigned a value that depends e.g. on the characteristics of the voxel environment. According to [8] three principal types of non-linear algorithms to estimate the transformation can be classified:

  - *Competitive* algorithms use an energy term that regularizes the transformation estimation. A typical example are the *elastic* registrations where the deformation is modeled as a physical process which resembles the stretching of elastic material such as rubber.
  - *Fluid* algorithms constrain the transformation to evolve in a continuous manner to arrive at the solution. These registration technics are utilized to model highly localized deformations.
  - *Parametric* algorithms work with parametric transformation representations like polynomials or splines.

  Most non-linear registration can be formulated as an optimization problem whose goal is to minimize an associated cost function like

  $$C = -C_{similarity} + C_{deformation}.$$

  $C_{similarity}$ stands for the similarity between the source and the target image whereas $C_{deformation}$ characterizes the cost function associated with the deformation which is often a measure for its smoothness.

CHAPTER 2

Analysis of the Problem Environment

In this thesis, we want to create a program that registrates two images with a *non-linear* transformation in a fully-automatical way. The program will use the outputs of a *block-matching* algorithm that is applied to the two images as input for its transformation estimation. Finally, the program will extend already existing program called `Baladin` (Epidaure Project, INRIA, Sophia Antipolis) which does a *linear* transformation estimation.

This chapter is subdivided in the following manner:

In section 2.1, the block-matching algorithm with all its options concerning the functional background and the similarity measures is described. In section 2.2, the capacities of the `Baladin` program as well as its execution in a multiscale approach are depicted.

Section 2.3 presents the non-linear registration idea and explains the realization of the non-linear transformation estimation using the block-matchinig algorithm.

2.1 The Block-Matching Algorithm

The block-matching algorithm is one of the two principal components that are successively executed to estimate the transformation here introduced. It works on the *iconic* features of the images. Prior to depicting its properties and functions in detail, below the block-matching algorithm is placed in its context regarding the registration algorithm:

Let I_S be our source and I_T be our target image volume. We search a transformation T that registrates I_S and I_T. The transformation T will be estimated with a non-linear algorithm (see section 2.3). The algorithm belongs to the so-called category *Pair-and-Smooth*, where two phases are distinguished that are executed alternately. The first phase works on the voxel intensity similarities (see section 2.3) to determine a correspondence whereas the second phase smoothes this result. To do so, we have to perform the following steps:

1. We identify corresponding point pairs $(\vec{x}_{I_S}, \vec{y}_{I_T})$ between I_S and I_T that form a *set of matches* or a *sparse displacement field*.

2. We calculate the transformation parameters based on the set of matches (see section 4.1).

The first step is realized by using a block-matching algorithm. Block-Matching algorithms find correspondences between *local* iconic features, therefore their appliance is well-suited for registrations where local deformations play a major role. The goal of applying the block-matching algorithm in the registration process is to receive a *sparse displacement field* as its output. The sparse displacement field is a set of matches where a number of point pairs that indicate the correspondences of voxels of the source image with voxels of the target image is stored. This information is used to estimate the image transformation T [9, 10].

2.1.1 Matching the Blocks

To obtain the sparse displacement field, we cut I_S into an ensemble of possibly overlapping subimages that we call *blocks*. Next, each block b is moved around in the volume I_T in order to find the position where its voxels hold the highest degree of intensity similarity with the voxels of I_T (see figure 2.1). (In section 2.1.2, the similarity measure will be discussed.) When that position is found, the coordinates of the centers $(\vec{x}_b, \vec{y}_b)$ of the respective corresponding blocks are stored as a *displacement vector* associated to the center position of the source image block.

Subsequently, the displacement vectors are used as sampling points to calculate the parametric transformation T so that for *each* source voxel the corresponding target voxel is determined.

To obtain the optimal result, there are 4 scheme parameters that can be modified so the algorithm works ideally according to the registration task at hand [2, 11, 12]:

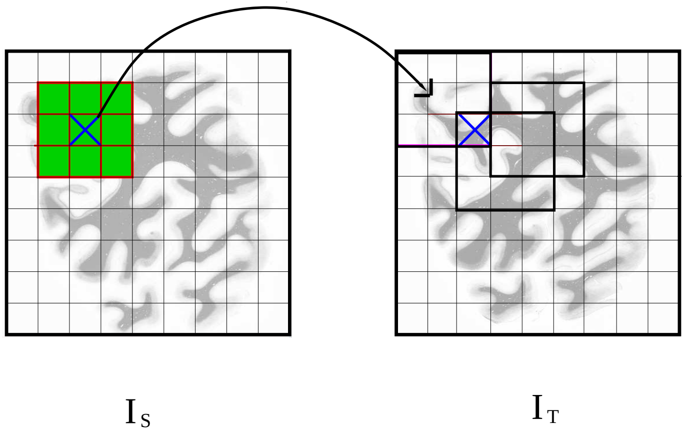


Figure 2.1: *The block-matching algorithm. For the grey shaded block in image I_S the position that features the greatest similarity in image I_T is searched. The search region is limited to a certain zone around the respective center coordinates of the block.*

- The size $\aleph$ of the blocks is defined as $\aleph = (\aleph_x, \aleph_y, \aleph_z)$. Taking into account the possible anisotropy of the image axes, $\aleph_x$,$\aleph_y$ and $\aleph_z$ can be assigned different values so not to weight one direction more than another. A big block size diminishes the influence of noise in the images but also the locality of the measures. Hence, a big block size reduces the fineness of details that will be retrieved.

- In most cases, the position of the target block that offers the highest similarity degree will be close to the respective position of the source block. Therefore, it is suggested to constrain the region of the correspondence research to a zone around the source block position. The size Ξ of this 'research neighbourhood' is defined as $\Xi = (2\Xi_x, 2\Xi_y, 2\Xi_z)$.

- According to the characteristics of the images to registrate, the sparse displacement field can be chosen to be more or less dense. We introduce to that effect the parameter $\Delta = (\Delta_x, \Delta_y, \Delta_z)$ that represents the distance between two consecutive blocks (the blocks may well overlap each other). $\Delta = (1, 1, 1)$ e.g. describes a vector density of one sampling point per voxel, however, to reduce the computing time, bigger distances are proposed.
  When there is no a priori knowledge about the image, the vectors should be uniformly distributed.

- The classical approach is to compare our source block to all of the blocks being contained in Ξ, that is, to do an *exhaustive* research in the neighbourhood. Though, on the assumption that the similarity measure obtained by comparing the block voxels is convex we can, for instance, examine just one voxel out of two and consider that the solution found by that method is close to the real solution. Then, $\Sigma = \Sigma_x, \Sigma_y, \Sigma_z$ represents the *resolution* of the sparse displacement field with $\Sigma_x, \Sigma_y, \Sigma_z$ describing the voxel distances between the research positions inside the neighbourhood. Choosing $\Sigma_x, \Sigma_y, \Sigma_z > 1$ reduces the computing time.

2.1.2 The Similarity Measure

The iconic criteria provide a measure for the similarity between the iconic features of the images regarding a certain position. The similarity is evaluated using a chosen *distance measure* comparing the voxel intensities of the images. The classical hypothesis used states that the distance measure diminishes in a reciprocally proportional manner with the quality of the registration.

To explain the different types of distance measures, we have to introduce some notations first: When performing a registration based on iconic features, *joint histograms* are a tool to determine the type of intensity relations. Let I_S and I_T be defined over the voxel grids Ω_S and Ω_T and let voxel $\vec{x}$ of I_S correspond to voxel $\vec{y}$ of I_T at the respective position. It holds:

$$\begin{aligned} I_S &: \quad \Omega_S \rightarrow \Lambda_S, \\ I_T &: \quad \Omega_T \rightarrow \Lambda_T. \end{aligned} \tag{2.1}$$

Here, Λ_S, Λ_T represent the ensembles of colors possibly found in the images. It is $\Lambda_S, \Lambda_T = [0, 255]$.

In a joint histogram $H(I_S, I_T)$ the probabilities that $I_S = i$ and $I_T = j$ with $i \in \Lambda_S$ and $j \in \Lambda_T$ are displayed (see figure 2.2). Given two identical images I_S and I_T, the joint histogram will show a perfect diagonal.

Considering the images as random variables, the values of the joint histogram can be interpreted like a probability density. The probability value p_{ij} expresses henceforth the probability that a randomly chosen voxel pair $(\vec{x}, \vec{y})$ will bear the intensity combination

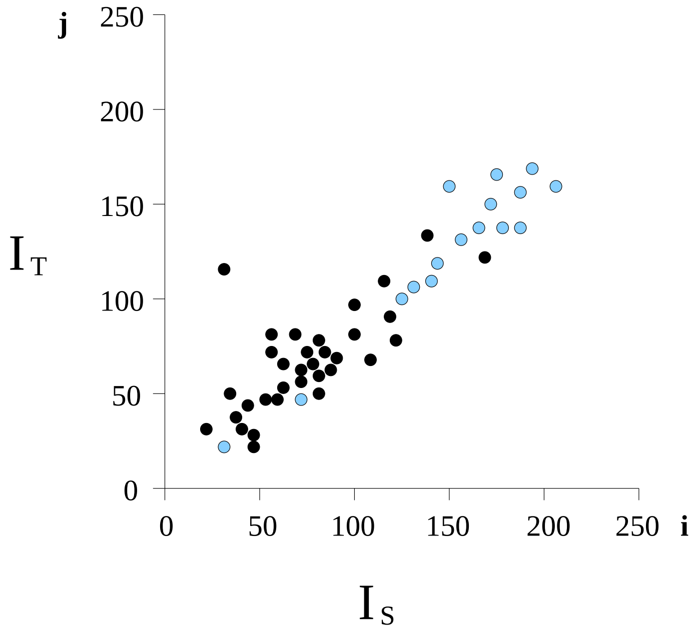


Figure 2.2: *Example for a joint histogram. The axes display the intensities of image I_S and image I_T in a grey scale. The number of voxel pairs with intensities (i, j) is represented with a circle, the darker the circle the greater the number.*

(i, j). It holds

$$p_{ij} = \frac{n_{ij}}{n} \tag{2.2}$$

with n being the sum of all voxel pairs and n_{ij} being the number of voxel pairs with intensities (i, j).

The choice of the distance measure depends on the intensity relations between the two images to be registrated. The simplest approach relies on the assumption that the only difference between the two images after the registration will be a Gaussian noise. In that case, the *Sum of Squared Differences (SSD$_H$)* or the *Sum of Absolute Differences (SAD$_H$)* are utilized to compare the transformed image $I_{S,new}$ with the target image I_T (see equations below). It is defined

$$SSD_H(I_{S,new}, I_T) = \sum_n p_{ij}(i_{\vec{x}} - j_{\vec{y}})^2,$$
$$SAD_H(I_{S,new}, I_T) = \sum_n p_{ij} \mid i_{\vec{x}} - j_{\vec{y}} \mid$$

where $i_{\vec{x}}$ and $j_{\vec{y}}$ are the intensities of the corresponding voxel positions $\vec{x}$ and $\vec{y}$ of the two images to be compared. The number of voxels is given with n. As can be seen, the intensity differences are weighted with p_{ij} which is a measure for the occurrence of an intensity combination (i, j) (see equation (2.2)). In most cases, we work with images that differ more than just by Gaussian noise, hence, we assume that an affine relation between the two images exists. Here, $i = \alpha j + \beta + \varepsilon$ holds denoting a linear relation with $\alpha, \beta \in \mathbb{R}$ and ε being a Gaussian noise. The distance measure utilized is called *correlation coefficient* ρ which is defined by

$$\rho^2(I_{S,new}, I_T) = \frac{Covariance(I_{S,new}, I_T)^2}{Variance(I_{S,new})Variance(I_T)} \tag{2.3}$$

where

$$Covariance(I_{S,new}, I_T) = \frac{1}{n}\sum_n (i_{\vec{x}} - \bar{I}_{S,new})(j_{\vec{y}} - \bar{I}_T)$$

and

$$Variance(I) = \frac{1}{n}\sum_n (i_{\vec{x}} - \bar{I})^2.$$

$\bar{I}_{S,new}$ and $\bar{I}_T$ denote the average intensity of the source and the target image respectively. In the `Baladin` program, the correlation coefficient is the distance measure of choice. In one block of the histological slices to be registrated mostly just two types of tissues are found, therefore, a linear relation between the intensities comprised in the blocks is probable [1].

2.2 The `Baladin` Program

The `Baladin` program realizes a three-dimensional registration method that is based on a local matching of iconic features combined with a global estimation of the transformation. `Baladin` registrates on voxel basis and fully automatically. The first version of the program (Aladin) was implemented at the Epidaure project, Inria, Sophia Antipolis in 2002, see [1], to facilitate the construction of a three-dimensional histological atlas by aligning its two-dimensional histological slices. The revised version now used is called `Baladin`.

2.2.1 Capacities of `Baladin`

To obtain a local matching of the iconic features of the image volumes I_S and I_T, `Baladin` applies the block-matching algorithm (see section 2.1) to the images, hence, a set of matches is created so that for a number of source voxels the corresponding target voxels are known. Those corresponding voxel position pairs $(\vec{x}_b, \vec{y}_b)$ are called *sampling points*. Thereupon, the transformation T is estimated that assigns *each* voxel of the source image a corresponding voxel of the target image. The desired transformation $\hat{T}$ smoothly extrapolates the correspondence to the whole space while minimizing the displacement error on the sampling points:

$$\hat{T} = \underset{\mathrm{T}}{argmin} \sum_b \rho(r_b).$$

The parameter $r_b \in \mathbb{R}^3$ describes the difference between the deformed source voxel of position $\vec{x}_b$ and its corresponding target voxel of position $\vec{y}_b$, thus, $r_b = T(\vec{x}_b) - \vec{y}_b$. The function $\rho(r) : \mathbb{R}^3 \rightarrow \mathbb{R}$ denotes the norm of that measure. If a least-squares approach is used then $\rho(r_b) = \|r_b\|_2^2$ holds. To ensure the robustness of the algorithm, a more elaborate idea utilizes a *weighted* least-squares estimation. Here, each displacement vector is furnished with a *weighting coefficient* (or *confidence value*) that regulates its influence in the search for the transformation. The value of the weight depends on the intensity variance in the target region around the original center of the respective source image block. A low variance presents a high risk to find an 'optimal' position that is incorrect because the blocks contained in the neighbourhood do not differ much in their similarity measures. Hence, a displacement vector belonging to a block showing a low variance is equipped with a small weight whereas a displacement vector situated in a region of high variance is weighted with a great value. The estimator implemented in the `Baladin` program is called least-trimmed-squares (LST) estimator. Here, the sampling points having values that differ too much from the average sampling point values are eliminated so that in the end only 60% of the sampling points computed by the block-matching algorithm are used as input for the transformation estimation.

The transformation classes implemented in the `Baladin` program to calculate T are linear, it can be chosen between a ***rigid*** and an ***affine*** approach, that is

$$\begin{aligned} T(\vec{x}) &= R\vec{x} + t \qquad \text{or} \\ T(\vec{x}) &= A_{sc}A_{sh}R\vec{x} + t \end{aligned}$$

as described in chapter 1.2.

To obtain an improved result, the block-matching algorithm and the subsequent transformation estimation are repeatedly applied to the two image volumes. In each new iteration step, the deformed source image of the last step serves as new source image (for further details see chapter 5) so that the estimation of the final transformation T consists of several transformations that are concatenated.

2.2.2 The Multiscale Approach

As described in section 2.1, there are different parameter values to be chosen in the block-matching algorithm to meet best the requirements of the application. An iteratively repeated exhaustive search of blocks of small sizes in big neighbourhoods would yield a high quality result, but the computational time may be unacceptably long. To avoid this and still gain the advantages of the iterating approach, `Baladin` realizes a *multiscale* approach that is based on a reduction algorithm, also known as *Gaussian and Laplacian pyramids*. The basic idea is introduced in the following:

While the representation of fine scales requires the full resolution, coarser scales can be represented at a lower resolution. A coarser scale image means that here a certain quantity of pixels of the full resolution image will be represented as just one pixel, this process is referred to as *subsampling*. It leads to a succession of images that become smaller and smaller with increasing coarseness of the scale, therefore, it is called a 'pyramid'. When subsampling an image, the 'Nyquist Theorem' has to be taken into account, that is, when we do not want to lose information by subsampling an image, the minimum sampling rate must be greater than the double of the maximal frequency of our image structures. To ensure this, an adapted smoothing filter must be applied to the image before sampling.

The classical approach is to fix the base size of the pyramid (= the original image) to a power of 2. When acting on the assumption that we utilize an image of size $X \times Y \times Z = 2^n \times 2^n \times 2^n$ the different levels i of the pyramid are obtained by dividing the respective

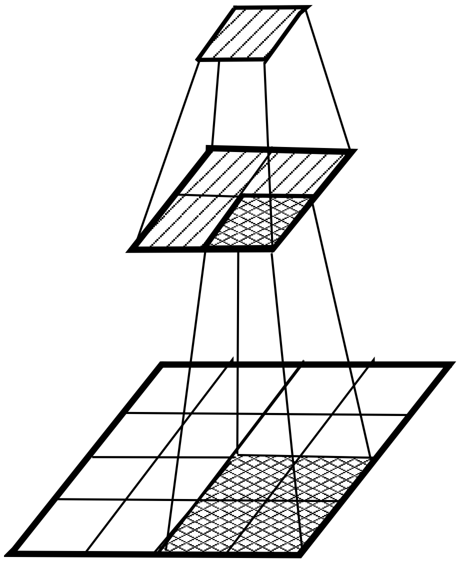


Figure 2.3: *2D quarternaire Gaussian pyramid. From one level to the next, the number of pixels is quadrupled. Each parent pixel possesses four children, each child possesses one parent.*

previous level size by 2. Therefore, the size $level\_size_l$ of level l is calculated like

$$level\_size_l = 2^{n-l} \times 2^{n-l} \times 2^{n-l}.$$

Hence, the pyramid is made out of $lmax = \log_2(X) + 1$ levels (see figure 2.3) [13].

In `Baladin`, the multiscale strategy is applied in the following way: The image is repeatedly subsampled so that a Gaussian pyramid is built until the image of level l reaches the same size as the chosen size $\aleph = (\aleph_x, \aleph_y, \aleph_z)$ of the blocks. On this image, the block-matching algorithm followed by a transformation estimation is applied iteratively until a satisfying result is achieved. In the next step, this transformation is approximated on the previous level $l - 1$ of the Gaussian pyramid, then the block-matching algorithm and the transformation estimation are applied and so forth. As the size of the blocks stays the same in each level, the *relative* size of the blocks compared to the image volume is decreased in each step, and the block-matching algorithm moves from being global at the beginning to being local at the end.

2.3 Non-Linear Registration Using Block-Matching

As mentioned in section 2.2.1, the transformation classes solving the registration problem that are implemented in `Baladin` are *linear* transformation classes. The linear regis-

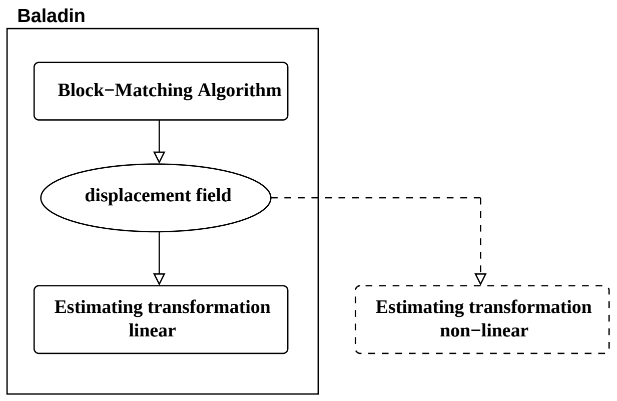

Baladin
Block−Matching Algorithm
displacement field
Estimating transformation linear
Estimating transformation non−linear

Figure 2.4: *The* `Baladin` *program is extended with a* non-linear *transformation class. The extension will use the same block-matching algorithm as* `Baladin` *to obtain the displacement field as input.*

tration method works well for e.g. registrating bone images of one modality that come from the same patient. However, images of the same structure that come from different patients will bear characteristics that cannot be registrated with an affine transformation alone as the variabilities of the features come into play. The features vary in a way that a linear function cannot represent. Here, we have to determine and apply a *non-linear* transformation. On this account, we want to extend the `Baladin` program with a new transformation class that is non-linear. We will present another way to work on the output of the block-matching algorithm, so to say the set of matches. By using the given displacement vectors as sampling points for an approximation, we will obtain a non-linear transformation estimation (see figure 2.4).

In order to find the transformation T that registrates the source image volume I_S with the target image volume I_T, we want to compute a displacement vector $\vec{u}$ for *each* voxel coordinate (x, y, z) of I_S:

Definition 2.1 *Transformation:*

$$\begin{aligned} T &: \mathbb{R}^3 \to \mathbb{R}^3, \\ T(\vec{x}) &= \vec{x} + \vec{u}(\vec{x}) = \vec{x}_{new}, \end{aligned} \tag{2.4}$$

$\vec{x} = (x, y, z)$: *3D coordinate of voxel position,*

$\vec{u}(\vec{x})$: *3D displacement vector of voxel at position* (x, y, z),

$\vec{x}_{new}$: *3D coordinate of the position of the transformed voxel.*

We work on basis of the block-matching algorithm as input to the transformation estimation. The block-matching algorithm is applied to I_S and I_T; its output consists of a set of N correspondent voxel pairs $(\vec{x}_b, \vec{y}_b)$ with $\vec{x}_b \in \Omega_S$, $\vec{y}_b \in \Omega_T$ and $b \in [1, N]$ (see section 2.1).
We look for a way to determine the transformation T by exploiting our knowledge about the ensemble of the N voxel pairs. They will serve as *sampling points* $\vec{u}_b(\vec{x}_b)$ for the displacement vectors $\vec{u}(\vec{x})$. It holds

$$\vec{u}_b(\vec{x}_b) = \vec{x}_b - \vec{y}_b. \tag{2.5}$$

A classical approach to approximate a function when several sampling points are known is to use *B-Splines*. They have mathematical advantages that renders them well adapted for this task as well as numerical advantages that makes them suitable to be employed in a program. In chapters 3 and 4.1 the numerical procedures based on B-Splines that are executed to find the desired transformation T are explained in detail.

CHAPTER 3

B-Splines

In the first part of this chapter, a definition of B-Spline functions is given followed by a description of their properties and an illustration of their characteristics. The second part deals with a specific type of B-Spline functions, the *cubic* B-Splines.

A spline function is a piecewise polynomial function that is used to interpolate or approximate curves or volumes of which a certain number of sampling points is known. Let $[a, b]$ be a given interval and let $t_0 < t_1 < ... < t_{n+k}$ be a given partition of $[a, b]$ with knots $t_i,\ \ i = 0, 1, ..., n + k.$ The associated normalized B-Spline basis function $N_{i,k}$ of order k is defined in a recursive way as described in the following.

Definition 3.1 *B-Spline basis function:*

$$\begin{aligned} \textit{for } k = 1: \qquad & N_{i,1}(t) = \begin{cases} 1, & \textit{for } t_i \le t < t_{i+1} \\ 0, & otherwise \end{cases} \\ \textit{for } k > 1: \qquad & N_{i,k}(t) = \frac{t - t_i}{t_{i+k-1} - t_i} N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t). \end{aligned} \tag{3.1}$$

B-Spline basis functions possess the characteristic traits depicted in the theorem below:

Theorem 3.1 *Properties of B-Spline basis functions (see [14], chapter 4):*

- *B-Spline basis functions have support* only on the interval $[t_i, t_{i+k}]$*:*

  $N_{i,k}(t) > 0$ *for* $t_i \leq t < t_{i+k}$,

  $N_{i,k} = 0$ *for* $t_0 \leq t < t_i$ *and* $t_{i+k} \leq t \leq t_{n+k}$.

- $\sum_{i=0}^{i=n} N_{i,k}(t) = 1$ *for* $t \in [t_{k-1}, t_{n+1}]$

- $N_{i,k}(t)$ *has continuity* C^{k-2} *at each of the knots* t_i.

- *For equally spaced knots we refer to* $N_{i,k}(t)$ *as* uniform *B-Splines, for non-equally spaced knots they are called* non-uniform *B-Splines.*

A B-Spline of order k is made out of B-Spline basis functions that are piecewise polynomials of degree $k-1$. If we impose a smooth continuity up to order $k-1$ of the spline and its derivatives at the knots, there exists only one degree of freedom per basis function [14].

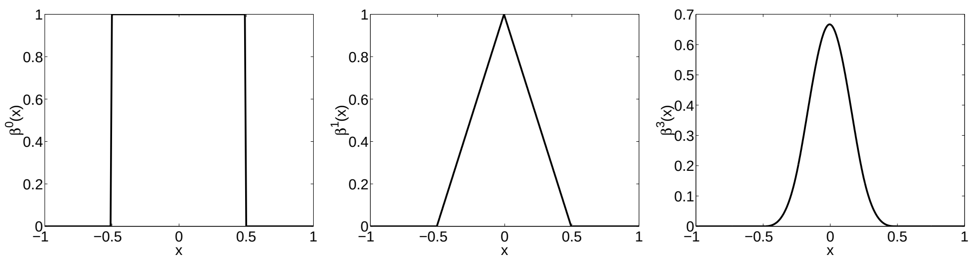
A descriptive non-recursive way to model B-Splines uses a convolution approach, thus, a B-Spline basis function of order k is constructed from the k-fold of a rectangular pulse β^0.

$$\begin{aligned} \beta^0(x) &= \begin{cases} 1, & -\frac{1}{2} < x < \frac{1}{2} \\ \frac{1}{2}, & \mid x \mid = \frac{1}{2} \\ 0, & otherwise \end{cases} \\ \beta^l(x) &= \underbrace{\beta^0 * \beta^0 * \ldots * \beta^0}_{l+1 \text{ times}}. \end{aligned} \tag{3.2}$$

$$\text{with } l = k-1 \text{ and } \beta^l = N_{k-1,i}$$

for an interval $[t_i, t_{i+1}]$ that is centered around zero.

To illustrate their appearance, the basis functions of order 1, 2, and 4 are displayed in figure 3.



(a) *B-Spline basis function of order* 1. (b) *B-Spline basis function of order* 2. (c) *B-Spline basis function of order* 4.

Figure 3.1: *B-Spline basis function of different orders.*

To create a B-Spline curve $X(t)$ that approximates a function $\mathbb{R} \to \mathbb{R}$ we use a superposition of the basis B-Spline functions of order k that were defined above.

Definition 3.2 *B-Spline curve in 1D:*

$$X(t) = \sum_{i=0}^{n+k} P_i N_{i,k}(t), \quad P_i \in \mathbb{R}. \tag{3.3}$$

The B-Spline curve $X(t)$ in the interval $[t_0, t_n]$ is composed of the sum of $n + k$ overlapping weighted B-Spline functions $N_{i,k}(t)$. Because of the local support of $N_{i,k}(t)$ *exactly* ***k*** B-Spline basis functions that are non-zero overlap in each interval $[t_i, t_{i+1}]$.
The points P_i associated with $N_{i,k}$ are called *control points* and serve as *B-Spline coefficients*. (In the following we will always refer to the B-Spline coefficients as control points.) To determine the B-Spline curve that approximates a given problem in the best way possible, the optimal value for each control point has to be found. By doing so, each piecewise polynomial B-Spline basis function in a B-Spline curve is equipped with its respective control point value as coefficient. Thus, all B-Spline curves are unambiguously characterized by their sequence of control points [15, 16].
Due to the fact that the basis functions have local support, each control point P_j influences $X(t)$ only for $t_j < t < t_{j+k}$. To ensure that all k basis functions are present in each interval considered, we have to choose $n \geq k-1$ and $t \in [t_{k-1}, t_{n+1}]$ [14]. If that was not the case, the $k-1$ outermost intervals would be equipped with less than k overlapping basis B-Spline functions.
A B-Spline curve X that approximates a function $\mathbb{R} \to \mathbb{R}^3$ looks like described in equa-

tion (3.3), except that now $P_i \in \mathbb{R}^3$ (instead of $P_i \in \mathbb{R}$) yields as the control points are distributed in the 3D space. However, to approximate a function $\mathbb{R}^3 \rightarrow \mathbb{R}^3$ in 3D with a B-Spline *volume* in 3D, equation 3.3 needs to be extended.

Definition 3.3 *B-Spline volume in 3D:*

$$X(x,y,z) = \sum_{o=0}^{n_x}\sum_{p=0}^{n_y}\sum_{q=0}^{n_z} N^x_{o,k}(x)N^y_{p,k}(y)N^z_{q,k}(z)P_{o,p,q}, \qquad P_{o,p,q} \in \mathbb{R}^3, \tag{3.4}$$

with n_x, n_y, and n_z being the numbers of intervals in x-, y- and, z-direction, o, p, q denoting the position of the control point and P giving the value of the control point.

The function $X(x,y,z)$ represents the three-dimensional B-Spline volume built by the 3D tensor product of the 1D B-Spline functions $N^x_{o,k}(x)$,$N^y_{p,k}(y)$ and $N^z_{q,k}(z)$ that are all of order k.
Each position (x,y,z) in the volume is assigned to a point $X(x,y,z)$ in 3D space, that is, the desired function is approximated in all three directions using three basis B-Splines N^x_o, N^p_j and N^z_q. Thus, the control points here are also elements of $\mathbb{R}^3$ and are arranged in a regular 3D grid.

As deduced from equation 3.4 the three-dimensional *cubic* B-Spline volumes are defined like this:

Definition 3.4 *Cubic B-Spline volume in 3D:*

$$X_{cubic}(x,y,z) = \sum_{o=0}^{n_x}\sum_{p=0}^{n_y}\sum_{q=0}^{n_z} N^x_{o,3}(x)N^y_{p,3}(y)N^z_{q,3}(z)P_{o,p,q}, \qquad P_{o,p,q} \in \mathbb{R}^3, \tag{3.5}$$

with n_x, n_y, and n_z being the numbers of intervals in x-, y- and, z-direction.
$N^x_{o,4}(x)N^y_{p,4}(y)$, and $N^z_{q,4}(z)$ are of order 4; they are called cubic *B-Spline basis functions.*

Theorem 3.2 *Properties of a cubic B-Spline volume:*

Two basis functions that meet at a knot

- *have the same value at the knot:* $N_{i,k}(t_{i+1}) = N_{i+1,k}(t_{i+1})$,
- *have an equal gradient at the knot:* $N'_{i,k}(t_{i+1}) = N'_{i+1,k}(t_{i+1})$,
- *have the same bending at the knot:* $N''_{i,k}(t_{i+1}) = N''_{i+1,k}(t_{i+1})$.

Therefore, the function X_{cubic} *describes a volume that is smooth because the basis functions come with a* C^2 *continuity at the knots.*

Their local support comprises 4 knots, thus, in an interval $[t_i, t_{i+1}]$ four basis B-Splines are superimposed as can be seen in figures 3.2 and 3.3.
In both figures, the length of one interval $[t_i, t_{i+1}]$ equals 1. According to what is said in the first part of this chapter the cubic B-Spline volume is composed of piecewise polynomials up to the degree of 3.

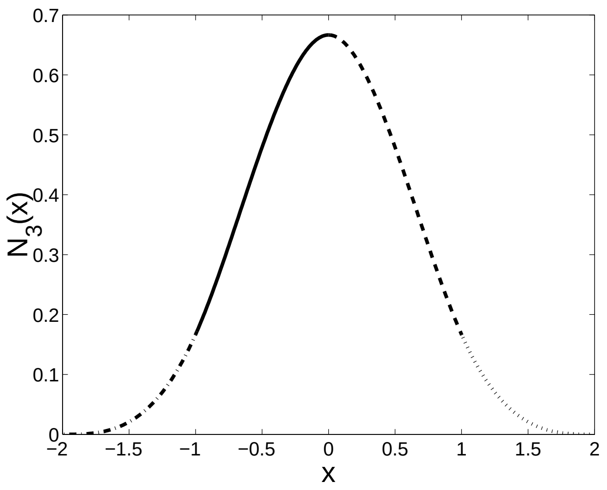


Figure 3.2: *One basis function of the cubic B-Spline. It comprises 4 intervals. In this example, the length of one interval is fi xed to 1.*

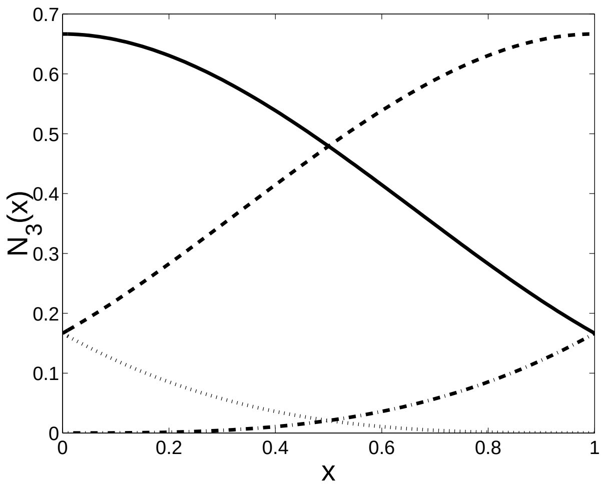


Figure 3.3: $k = 4$ *overlapping cubic B-Spline basis functions in one interval. In this example, the length of the interval is fi xed to 1.*

CHAPTER 4

3D B-Spline Estimation

As described in chapter 1 and section 2.3, the topic of this thesis is the estimation of a non-linear transformation to registrate the source image I_S with the target image I_T. Using the block-matching algorithm as explained in section 2.1 will give us a set of correspondences between the two images..

The 3D B-Spline estimation introduced here will use the set of correspondences as sampling points upon which it approximates the transformation function with a *least squares* approach.

This chapter is subdivided as follows:

Section 4.1 explicates how the B-Spline functions are used to represent the desired transformation.

In section 4.2 the application of a least squares estimator to calculate the transformation on basis of the B-Spline functions is explained.

Section 4.3 deals with the introduction of *weighted sampling points* to the approximation algorithm whereas section 4.4 proposes the extension of the algorithm with a *regularization term*.

In section 4.5 the approach of a *multiresolution strategy* concerning the choice of the number of control points is presented.

4.1 Representing the Transformation using B-Splines

In this thesis, we employ the transformation class of uniform B-Splines to approximate $\vec{u}(x, y, z)$. We want to transform an image volume, consequently, $\vec{u}(x, y, z) = \vec{u}_{BSplines}(x, y, z)$

is represented using equation 3.4 that describes B-Spline functions in 3D (see equation (4.2)).

Definition 4.1 *Representation of the cubic B-Spline approximation, version 1:*

$$\vec{u}_{BSplines}(x,y,z) = \sum_{i=0}^{n_x}\sum_{j=0}^{n_y}\sum_{k=0}^{n_z} N_i^x(x)N_j^y(y)N_k^z(z)P_{i,j,k}, \tag{4.1}$$

$$P_{i,j,k},\ \vec{u}_{BSplines}(x,y,z) \ \in\ \mathbb{R}^3$$

Moreover, we want to benefit from all advantages that are given by basis B-Splines of order 4 (see section 3), thus, we choose our B-Splines to be cubic ones which implies a order k of 4.

The number $n_x n_y n_z$ of control points for the spline approximation should be less than the number N of our sampling points. Instead of using *each* sampling point as a control point we introduce a control point grid coarser than the density of sampling points, hence, we do a B-Spline-*approximation* that yields the following advantages:

- Our number of unknowns is smaller, thus, we work with less degrees of freedom.
- We can use a multiresolution strategy that enables us to adaptedly increase the number of control points.
- Noise is reduced by using an approximative approach.

As pointed out in chapter 3, in each interval of length $\overline{t_i t_{i+1}}$ exactly k non-zero B-Spline basis functions overlap. Hence, equation (4.2) can be simplified; instead of regarding all n_x (n_y, n_z) intervals in every sum we now just calculate the sum of 4 values in the respective interval that we consider to have the length 1. For each B-Spline sum we need 4 control points in x-, 4 control points in y- and 4 control points in z-direction, thus, we consider 64 control points per coordinate (x_b, y_b, z_b). The representation therefore looks like depicted in equation (4.2), here B_l stands for $N_{l,4}$ with $[t_l, t_{l+1}] = [0,1]$, so the B in the equation denote *cubic* B-Splines:

Definition 4.2 *Representation of the cubic B-Spline approximation, version 2:*

$$\vec{u}_{BSplines}(x,y,z) = \sum_{l=0}^{3}\sum_{m=0}^{3}\sum_{n=0}^{3} B_l^x(\alpha)B_m^y(\beta)B_n^z(\gamma)P_{i+l,j+m,k+n}, \qquad P_{i,j,k} \in \mathbb{R}^3 \tag{4.2}$$

i, j, k :	position of the nodes of the control point grid
$P_{i+l,j+m,k+n}$:	values at the nodes of the control point grid
$B_l^x(\alpha), B_m^y(\beta), B_n^z(\gamma)$:	basis functions of the cubic B-Splines

As we deal now with B-Spline basis functions of length 1, their inputs that depend on the voxel positions $\vec{x} = (x, y, z)$ are determined using the following formulas:

$$\begin{aligned} \alpha &= \tfrac{x}{\Delta_x} - \lfloor \tfrac{x}{\Delta_x} \rfloor \\ \beta &= \tfrac{y}{\Delta_y} - \lfloor \tfrac{y}{\Delta_y} \rfloor \\ \gamma &= \tfrac{z}{\Delta_z} - \lfloor \tfrac{z}{\Delta_z} \rfloor \end{aligned}$$

To find the associated 3D interval with its $4+4+4$ B-Spline functions to a given voxel position (x, y, z), the neighbouring control point nodes are calculated like this:

Δ_x :	distance between the control points in x-direction
Δ_y :	distance between the control points in y-direction
Δ_z :	distance between the control points in z-direction
$i = \lfloor \frac{x}{\Delta_x} \rfloor - 1$:	to find the i closest to x
$j = \lfloor \frac{y}{\Delta_y} \rfloor - 1$:	to find the j closest to y
$k = \lfloor \frac{z}{\Delta_z} \rfloor - 1$:	to find the k closest to z

Therewith, we have obtained a parametric representation of the unknown transformation component $\vec{u}_{BSplines}(x, y, z)$.

4.2 3D B-Spline Least Squares Estimation

Our task now is to find the values for the parameters $P_{i,j,k}$ that render the optimal approximation of the transformation T *with respect to the ensemble of N sampling points*

$\vec{u}_b(\vec{x}) = \vec{u}_b(x, y, z)$ *given by the set of matches at hand*. We choose a least squares estimation as the criterion to be minimized (see equation (4.3)) which gives a linear problem [19]:

Definition 4.3 *Least squares estimation:*

Minimize the criterion

$$C = \sum_{s=0}^{N-1} \|\vec{u}_b(\vec{x}_b) - \sum_{l=0}^{3}\sum_{m=0}^{3}\sum_{n=0}^{3} B_l^x(\alpha) B_m^y(\beta) B_n^z(\gamma) P_{i+l,j+m,k+n})\|^2 \quad ! \tag{4.3}$$

By decomposing the norm into three sums along its x-, y-, and z-direction we obtain

$$C = S^x + S^y + S^z.$$

As we deal with a tensor product, S^x, S^y, and S^z rely on different parameters, thus, they can be minimized independently. Therefore, we deal now with the following three least squares estimators (see equations 4.4):

$$\begin{aligned} S^x &= \sum_{s=0}^{N-1} (u^x(\vec{x}_b) - \sum_{l=0}^{3}\sum_{m=0}^{3}\sum_{n=0}^{3} B_l^x(\alpha) B_m^y(\beta) B_n^z(\gamma) P^x_{i+l,j+m,k+n})^2, \\ S^y &= \sum_{s=0}^{N-1} (u^y(\vec{x}_b) - \sum_{l=0}^{3}\sum_{m=0}^{3}\sum_{n=0}^{3} B_l^x(\alpha) B_m^y(\beta) B_n^z(\gamma) P^y_{i+l,j+m,k+n})^2, \\ S^z &= \sum_{s=0}^{N-1} (u^z(\vec{x}_b) - \sum_{l=0}^{3}\sum_{m=0}^{3}\sum_{n=0}^{3} B_l^x(\alpha) B_m^y(\beta) B_n^z(\gamma) P^z_{i+l,j+m,k+n})^2. \end{aligned} \tag{4.4}$$

Here, $u^x(\vec{x}_b)$ stands for the x-component of vector $\vec{u}_b(\vec{x}_b)$, $u^y(\vec{x}_b)$ for the y-component and $u^z(\vec{x}_b)$ for the z-component.
The same notion applies for $P^x_{i+l,j+m,k+n}$, $P^y_{i+l,j+m,k+n}$ and $P^z_{i+l,j+m,k+n}$ so that

$$\begin{aligned} & u^x(\vec{x}_b), u^y(\vec{x}_b), u^z(\vec{x}_b) \in \mathbb{R} \\ \text{and} \quad & P^x_{i+l,j+m,k+n}, P^y_{i+l,j+m,k+n}, P^z_{i+l,j+m,k+n} \in \mathbb{R} \end{aligned}$$

holds.

We want to solve $S^x \overset{!}{\rightarrow} min$, $S^y \overset{!}{\rightarrow} min$ and $S^z \overset{!}{\rightarrow} min$ by finding the best adapted values for $P_{i,j,k}$. In the following, we consider only the solution to minimizing S^x, the proceeding for the two associated minimizations of S^y and S^z will be alike. Representing that issue in a matrix-vector approach leads to the following equation (4.5):

$$||\vec{u} - B\vec{p}||^2 \overset{!}{\rightarrow} min.$$

Here, $\vec{u}$ is composed of all N x-components $u^x(\vec{x}_b)$ of $\vec{u}_b(\vec{x}_b)$. All $n_x n_y n_z$ components of $P^x_{i,j,k}$ are comprised in $\vec{p}$ (a more detailed definition follows below). As the number N of the sampling points $\vec{u}_b$ should be greater than the number $n_x n_y n_z$ of control points $P_{i,j,k}$, we deal with a linear equation system that is overdetermined.

That fact serves to augment the accuracy of the least squares algorithm. Furthermore, it guarantees that an approximate solution exists. To find the best values for the control points we derive $||\vec{u} - B\vec{p}||^2$ with respect to $\vec{p}$ and set the derivation to zero.

$$||\vec{u} - B\vec{p}||^2 = \vec{u}^T\vec{u} - 2\vec{u}^T B\vec{p} + \vec{p}^T B^T B\vec{p}$$

$$\begin{aligned} &\tfrac{\delta}{\delta\vec{p}}(\vec{u}^T\vec{u} - 2\vec{u}^T B\vec{p} + \vec{p}^T B^T B\vec{p}) = 0 \\ \Leftrightarrow\quad & -2\vec{u}^T B + 2\vec{p}^T B^T B &= 0 \\ \Leftrightarrow\quad & \vec{p}^T B^T B &= \vec{u}^T B \\ \Leftrightarrow\quad & B^T B\vec{p} &= B^T\vec{u} \\ \Leftrightarrow\quad & \vec{p} &= (B^T B)^{-1} B^T \vec{u} \end{aligned} \tag{4.5}$$

with $\vec{u} \in \mathbb{R}^N, \quad \vec{p} \in \mathbb{R}^m, \quad B \in \mathbb{R}^{m\times N},$

N :	number of sampling points $\vec{u}_b$,
m :	number of control points $m = n_x n_y n_z$

The last step assumes that $B^T B$ has full rank, i.e. that the system is overdetermined.

To put it more efficiently from a computational point of view, we have to solve the equation

$$\begin{aligned} &M\vec{p} = \vec{v} \qquad (4.6) \\ &\text{with } M = B^T B \quad \in \mathbb{R}^{m \times m} \\ &\text{and } \vec{v} = B^T \vec{u} \quad \in \mathbb{R}^m \end{aligned}$$

Below, the components of the matrix-vector representation are described in a detailed way: The three-dimensional sampling points $\vec{u}_b(\vec{x}_b)$ given by our set of matches are arranged in three vectors with $\vec{u^x}$ containing the x-values of $\vec{u}_b(\vec{x}_b)$ etc.:

$$\vec{u}^x = \begin{pmatrix} u_0^x \\ u_1^x \\ . \\ . \\ . \\ u_{N-1}^x \end{pmatrix}, \quad \vec{u}^y = \begin{pmatrix} u_0^y \\ u_1^y \\ . \\ . \\ . \\ u_{N-1}^y \end{pmatrix}, \quad \vec{u}^z = \begin{pmatrix} u_0^z \\ u_1^z \\ . \\ . \\ . \\ u_{N-1}^z \end{pmatrix}.$$

Arranging the x-components of the control point values $P_{i,j,k}$ of our grid row by row in a vector we obtain $\vec{p}$ that is outlined below:

$$\vec{p} = \begin{pmatrix} p_{0,0,0} \\ p_{1,0,0} \\ . \\ . \\ p_{n_x-1,0,0} \\ p_{0,1,0} \\ . \\ . \\ . \\ p_{n_x-1,n_y-1,n_z-1} \end{pmatrix}$$

Here, n_x, n_y and n_z represent the number of control points in x-, y- and z-direction.

The matrix B of the system contains the B-Spline values necessary to set up the triple sum

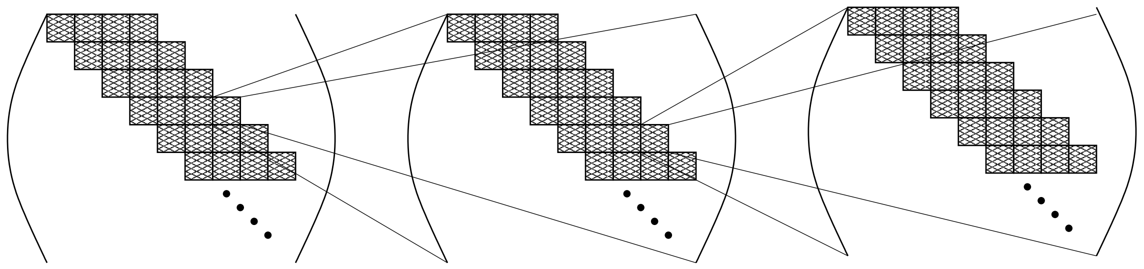


Figure 4.1: *Appearance of the B-Matrix when cubic B-Splines are used. Each shaded quadrate represents one entry of a row and several entries of a column (number depends on the number ratio sampling points/control points).*

equation (4.4), thus, the first row is associated with the sum for $s = 0$, the second for $s = 1$ and so forth. As each sampling point influences 64 control points in its neighbourhood, each row of the matrix contains 64 entries that are non-zero. The matrix can be arranged to have a diagonal appearance like depicted in figure (4.1) (compare [20]). Here, each square stands for a non-zero entry in one column and in several rows where the number of rows with the same structure depends on the 'sampling point - control point' ratio.
$B^T B$ is symmetrical positive in general, and definite positive when at least *one* sampling point appears in each of the three-dimensional intervals.

4.3 Weighted 3D B-Spline Least Squares Estimation

In this section, the idea of improving the robustness of the B-Spline estimation by extending the least squares approach with a *weighted* least squares approach is presented. Firstly, the weighted least squares approach is defined. Subsequently, its application to the B-Spline estimation with the adaptions necessary is explained.

As we aspire toward a robust algorithm, we have to establish a means to ensure that the calculation of the transformation T does not become instable due to partially incorrect data. To cover the fact that not all sampling points $\vec{u}_b(x_b)$ have the same significance we introduce a weighting factor w with $0 \leq w \leq 1$ to each difference in equation (4.4). In case of a high significance of $\vec{u}_b(\vec{x}_b)$, the value associated to w_b will be close to 1, in case of a low significance, w_b is given a value close to 0. By doing so, we ensure that outliers and sampling points that are not significant 'enough' loose influence on the resulting transformation.

Equation 4.7 below represents the weighted least-squares estimation of the transformation T in x-direction.

Definition 4.4 *Weighted least squares estimation:*
Minimize

$$S^x = \sum_b \|w_b(u^x(\vec{x}_b) - \sum_{l=0}^{3}\sum_{m=0}^{3}\sum_{n=0}^{3} B_l^x(\alpha)B_m^y(\beta)B_n^z(\gamma)P^x_{i+l,j+m,k+n})\|^2 \quad ! \tag{4.7}$$

The matrix representation looks like this:

$$B^TW^TWB\vec{p} = B^TW^TW\vec{u} \tag{4.8}$$
$$\text{with } W \in \mathbb{R}^{N\times N},$$

where W is a diagonal matrix that in each row contains the weight that corresponds to the element in the respective row in $\vec{u}$.
We have to solve

$$M\vec{p} = \vec{v} \tag{4.9}$$
$$\text{with } M = B^TW^TWB \quad \in \mathbb{R}^{m\times m}$$
$$\text{and } \vec{v} = B^TW^TW\vec{u} \quad \in \mathbb{R}^{m}.$$

The value of the weight w_b depends on the variance in the region of the respective block b (see section 2.2.1). A high variance augments the probability of the result being accurate. Therefore, a sampling point $\vec{u}_b(\vec{x}_b)$ is equipped with a great w_b if it lies in a high variance region and vice versa.

As described in chapter 3, B-Splines come with a local support. For our purpose, this property is ambivalent. On the one hand, the local support turns out to be a great advantage concerning sampling points that are significant outliers. In that case, the influence of the outlier is limited on a certain region of the transformation, therefore, the computation gains robustness. On the other hand, the combination of representing the wanted transformation T by B-Splines and simultaneously introducing weights to the least squares estimation (see section 4.3) bears the problem that some control points may be associated with the weight zero. That happens for example in a largely homogeneous region if all sampling points $\vec{u}_b(\vec{x}_b)$ of the local support of one specific control points $P_{i,j,k}$ are assigned the weight $w =$ zero. In that case, the matrix M mutates to a non-invertible matrix and the equation system becomes singular.

To find a solution to this problem, three different approaches are conceivable:

Firstly, we could prohibit the weight $w = 0$ altogether but choose a value that is close to zero instead for sampling points with a low significance. Here, the equation system will not become singular but the matrix M will be ill-conditioned.

Next, we could assess that for each control point at least a certain number of sampling points must have a value different from zero. That approach will surely prevent the matrix M from becoming ill-conditioned but it seems rather elaborate to be realized.

Finally, the problem might be solved by adding a regularization term (see section 4.4) to our criterion to be minimized as that measure will render the matrix much more well-conditioned.

4.4 The Regularization Term

In order to obtain a transformation that is smooth we introduce a penalty term to forestall resulting functions that are discontinuous. Hence, we formulate the registration task as an optimisation problem with the goal to minimize an associated cost function:

Definition 4.5 *Cost function in the transformation estimation:*

$$C = -C_{similarity} + \lambda C_{regularization} \tag{4.10}$$

Here, $C_{similarity}$ characterizes the similarity between the source and the target image, and $C_{regularization}$ represents the degree of discontinuity of the associated transformation function [17]. The parameter λ defines the tradeoff between the alignment of the two images and the smoothness of the transformation.

$C_{similarity}$ corresponds to the least squares estimation as seen in equation (4.6). As proposed in [18], we use a biharmonic model as depicted in equation (4.11) to describe the penalty term. The physical interpretation of the biharmonic model is the idea of approximating the energy of a thin plate of metal which is subjected to bending deformation [17].

Definition 4.6 *The regularization term:*

$$C_{regularization} = \frac{1}{V}\int_0^X\int_0^Y\int_0^Z\left[\left(\frac{\partial^2 T(\vec{p})}{\partial x^2}\right)+\left(\frac{\partial^2 T(\vec{p})}{\partial y^2}\right)+\left(\frac{\partial^2 T(\vec{p})}{\partial z^2}\right)+ 2\left(\frac{\partial^2 T(\vec{p})}{\partial xy}\right)+2\left(\frac{\partial^2 T(\vec{p})}{\partial xz}\right)+2\left(\frac{\partial^2 T(\vec{p})}{\partial yz}\right)\right]dxdydz. \tag{4.11}$$

To minimize C, we derive equation (4.10) with respect to $\vec{p}$:

$$\frac{\partial C}{\partial \vec{p}} = -\frac{\partial C_{similarity}}{\partial \vec{p}} + \lambda\frac{\partial C_{regularization}}{\partial \vec{p}} = 0. \tag{4.12}$$

4.5 The Multiresolution Strategy

By modifying the number of control points we are able to vary the performance of the algorithm as the number of control points determines the number of freedom degrees of the calculation. A small number of control points results in a transformation that is coarse but quickly computed whereas a large number of control points increases the computation time but produces a more refined registration result. To take advantage of these correlations, we employ a *multiresolution strategy* that comprises an iteration of the transformation iteration. Thus, in the first iteration $r = 1$ we begin with a small number of control points to receive a rough transformation T_1 that maps I_S to I_T. (The transformation calculated in each step r is denoted T_r.) By applying T_1 to I_S, we deform I_S to a new image $I_{S,new}$.

In the next iteration $r = 2$, we insert additional control points and repeat the calculation of the transformation. This time we use the deformation information stored in $I_{S,new}$ to calculate the next transformation $T_{r+1} = T_2$ (for the exact computational approach see chapter 5). We iterate until the desired refinement of the transformation is reached.

By applying the multiresolution strategy we receive a registration that is global at the beginning and becomes more and more local with every step.

Now we have to decide in which manner the additional control points will be inserted. One possibility consists in keeping our regular grid and inserting one additional control point in between each two already existing ones so that the grid resolution is doubled. Another possible idea is to insert additional control points depending on where the transformation needs to be refined. However, that approach causes the problem that *uniform* B-Splines can not be used any longer.

CHAPTER 5

Derivation of the Non-Linear Transformation

In this chapter, the details of derivating the desired transformation estimation are explained. The first section describes the recursive algorithm with the steps necessary to compute the transformation iteratively. The second section illustrates how the transformation calculation depicted in the first section is integrated in a multiscale approach (see also section 2.2.2).

In order to find the non-linear transformation T that registrates the source image I_S with the target image I_T, we compute a displacement vector $\vec{u}$ for *each* voxel coordinate $\vec{x}$ of I_S as defined in section 2.3, see equation (2.4):

$$T(\vec{x}) = \vec{x} + \vec{u}(\vec{x})$$

We start the calculation by applying the block-matching algorithm to the two images I_S and I_T to obtain a sparse field of point correspondences $(\vec{x}_b, \vec{y}_b)$ where $\vec{x}_b \in \Omega_S$ and $\vec{y}_b \in \Omega_T$ with Ω_S and Ω_T being the grid of voxels of I_S and I_T respectively. They form a displacement vector $\vec{u}_b(\vec{x}_b)$ that is assigned to each block b in the source image such that

$$\vec{y}_b = T(\vec{x}_b) = \vec{x}_b + \vec{u}_b(\vec{x}_b),$$

(see section 2.1).

To calculate T, we have to approximate $\vec{u}(\vec{x}),\ \ \vec{u} : \mathbb{R}^3 \to \mathbb{R}^3$, using the point pairs $(\vec{x}_b, \vec{y}_b)$ as sampling points. T shall deform I_S in a way that its features resemble the features of I_T best possibly.

As can be seen in the layout below, the transformation T does not change the intensities

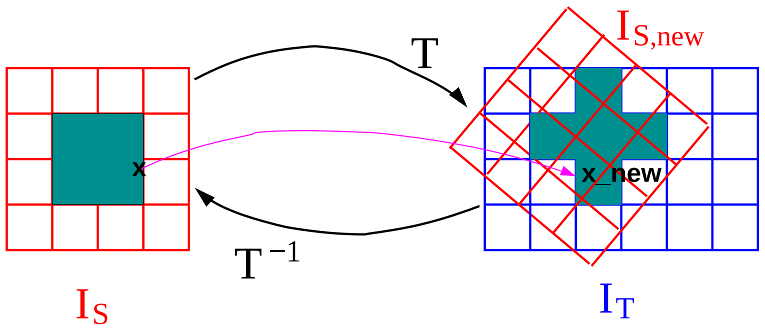


Figure 5.1: *Illustration of the Transformation T. The transformation T is applied to image I_S to obtain image $I_{S,new}$ that resembles I_T. With the transformation T^{-1} the voxels $\vec{x}$ associated to $\vec{x}_{new}$ are determined by $\vec{x} = T^{-1}(\vec{x}_{new})$.*

but the positions of the voxels $\vec{x}$:

$$\left| \begin{array}{l} \vec{x} \\ I(\vec{x}) \end{array} \right. \quad \xrightarrow{T} \quad \left| \begin{array}{l} T(\vec{x}) = \vec{x}_{new} \\ I_{S,new}(\vec{x}_{new}) = I(\vec{x}) \end{array} \right.$$

Definition 5.1 *Transformation Application:*

$$\begin{aligned} & I_{S,new} := T \star I_S, \\ \textit{with} \qquad & I_{S,new}(T(\vec{x})) = I_S(\vec{x}), \end{aligned} \tag{5.1}$$

where $\star$ denotes the application of a transformation T on an image I [6].

$T(\vec{x})$ represents the new position $\vec{x}_{new}$ for the voxel at position $\vec{x}$, hence, T applied to I_S deforms I_S so that the result will be the new image $I_{S,new}$ (see figure 5.1). When we apply the transformation T to I_S, the new image $I_{S,new}$ comprises the new coordinates $\vec{x}_{new}$ that are associated to each source voxel position $\vec{x}$ (see equation (5.2) below).

$$T \star I_S(\vec{x}) = I_{S,new}(T(\vec{x})) = I_{S,new}(\vec{x}_{new}) \tag{5.2}$$

To determine which *intensity* in $I_{S,new}$ is associated to the voxel at position $\vec{x}_{new}$, we have to find the place where the voxel came from and, therefore, detect the corresponding position $\vec{x}$ in I_S (see figure 5.2). Thus, we apply the inverse transformation T^{-1} to $I_{S,new}$

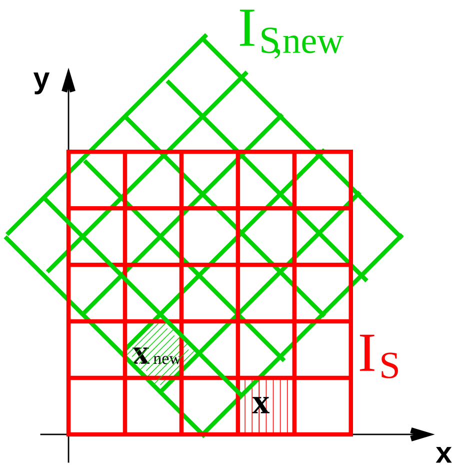


Figure 5.2: *Detecting the corresponding positions between I_S and $I_{S,new}$. To determine the intensity of $\vec{x}_{new}$ in $I_{S,new}$ the associated $\vec{x}$ in I_S must be detected.*

and get

$$\begin{aligned} \vec{x} &= T^{-1}(\vec{x}_{new}) \quad \text{and} \\ I_{S,new}(\vec{x}_{new}) &= I_S(T^{-1}(\vec{x}_{new})). \end{aligned} \tag{5.3}$$

Hence, we have to model the transformation

$$T^{-1}(\vec{x}_{new}) = \vec{x}_{new} + \vec{u}^{[-1]}(\vec{x}_{new}) \tag{5.4}$$

where $\vec{u}^{[-1]}(\vec{x}_{new})$ denotes the negative vector to $\vec{u}(\vec{x})$:

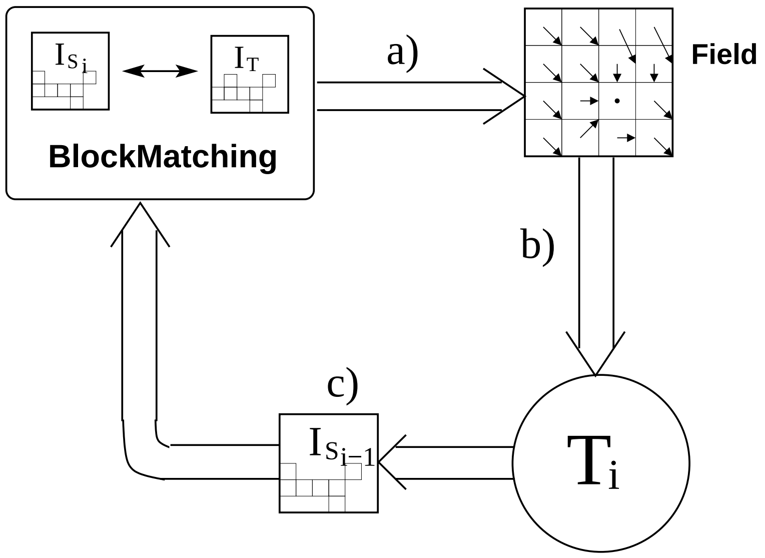
$$\vec{u}^{[-1]}(\vec{x}_{new}) = -\vec{u}(\vec{x}).$$

Note: $\vec{u}$ and $\vec{u}^{[-1]}$ are not inverse referring to the operator $\star$ as

$$\vec{u}^{[-1]}(\vec{u}(\vec{x})) \neq \vec{x}.$$

We will represent $\vec{u}^{[-1]}(\vec{x}_{new})$ with a B-Spline approximation as described extensively in chapter 4.

We chose to determine the transformation by using cubic B-Splines (see chapter 3 and sections 4.1) and 4.2.



a) The result of the block-matching algorithm is a field of displacement vectors.
b) The sparse displacement field is used to calculate the transformation T_i
c) T_i is applied to the source image (see also section 5.1).

Figure 5.3: *Iterated estimation of T.*

5.1 Calculation of the Transformation Iteration

The transformation is computed **iteratively**. At each iteration a transformation T_i^{-1} is calculated. T_i^{-1} is then applied to the source image so that the respective next iteration incorporates the results found before. The estimation procedure has to undergo the same operational steps repeatedly. This cyclic context is simplifiedly illustrated in figure 5.3. In the following, those steps are explained in a detailed manner:

Initialisation of T_i:

We perform the first iteration $i = 1$ on our way to find the transformation that matches best possibly the voxels of our source with our target image. The following moves have to be made:

- We use the source image and the target image as input to the block-matching algorithm of the `Baladin` program (see section 2.2). We retrieve a set of matches as the result that contains N corresponding voxels $(\vec{x}_b, \vec{y}_b)$ with $b = 1, ..., N$. Those point pairs indicate the displacements $\vec{u}_b(\vec{x}_b)$ with

$$\vec{u}_b(\vec{x}_b) = \vec{y}_b - \vec{x}_b \quad \text{with} \quad \vec{u}_b : \mathbb{Z}^3 \rightarrow \mathbb{Z}^3.$$

Hence, voxel $\vec{x}_b$ of image I_S corresponds to voxel $\vec{y}_b = \vec{x}_b + \vec{u}_b(\vec{x}_b)$ of image I_T.

To approximate the transformation, we have to adapt the resulting $\vec{u}_b(\vec{x}_b)$ to the search for T_i^{-1} (see equation (5.4)). They must be inverted, so

$$\vec{u}_b^{[-1]}(\vec{y}_b) = -\vec{u}_b(\vec{x}_b)$$

holds.

- Based on the sampling points $\vec{u}_b^{[-1]}(\vec{y}_b)$ we compute $u_i^{[-1]}(\vec{x}_{new})$ in each iteration i with $u_i^{[-1]}(\vec{x}_{new}) : \mathbb{R}^3 \to \mathbb{R}^3$. We use the B-Spline approximation and determine therewith $T_i^{-1} = T_1^{-1}$.
- We resample I_S: $I_{S,new}(\vec{x}_{new}) = I_{S,i}(\vec{x}_{new}) = I_S(T_1^{-1}(\vec{x}_{new}))$.

Calculation of T_{i+1}:

In a recursive approach, we would estimate a residual transformation δT_{i+1} from the deformed source image $I_{S,i}$ of iteration i to the deformed source image $I_{S,i+1}$ of iteration $i+1$ to calculate the transformation T_{i+1}. However, the action of resampling an image brings along a loss of data which would accumulate with each iteration during an recursive process. To avoid this, we do not want to compose recursively the image resamplings. Instead, we need to compute T_{i+1} from I_S straight to the deformed source image $I_{S,i+1}$ of iteration $i+1$. Hence, we do not resample $I_{S,new} = I_{S,i}$ into $I_{S,i+1}$ but we resample I_S directly into $I_{S,i+1}$.

To do so, we exploit the information being enclosed in the transformed image $I_{S,i}$ in the following way:

Using $I_{S,i}$ we determine point pairs between the original source image I_S and I_T that correspond better than the sampling points used for the iteration before. Those point pairs will serve as sampling points $\vec{u}_b(\vec{x}_b)$ for the calculation of T_{i+1} (see figure 5.4). Below, the measures taken to find the newly corresponding point pairs $(\vec{x}_{new,b}, \vec{y}_b)$ are explained:

- We compute the matches from $I_{S,new} = I_{S,i}$ to I_T by applying the block-matching algorithm.

  As a result, we obtain the corresponding voxel pairs $(\vec{x}_{new,b}, \vec{y}_b)$ with $\vec{x}_{new,b} \in \Omega_{S,new}, \ \ \vec{y}_b \in \Omega_T$.

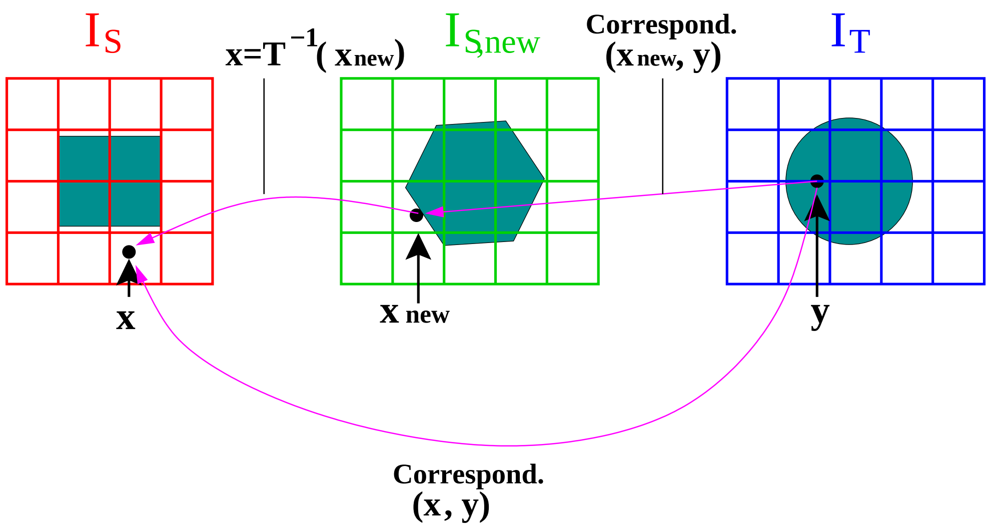


Figure 5.4: *Finding the corresponding points between I_S and I_T (simplified representation). The image $I_{S,new}$ serves to find a better correspondence between I_S and I_T than in the iteration before. The point pair (x_{new}, y) is calculated by the block-matching algorithm applied to $I_{S,new}$ and I_T.*

- For all point pairs $(\vec{x}_{new,b}, \vec{y}_b)$ that establish the correspondences between $I_{S,new}$ and I_T we find the respective corresponding points $\vec{x}_b$ in I_S by calculating

$$\begin{aligned} \vec{x}_b &= T_i^{-1}(\vec{x}_{new,b}) \\ &\overset{(2.4)}{=} \vec{x}_{new,b} + \vec{u}_i^{[-1]}(\vec{x}_{new,b}). \end{aligned} \tag{5.5}$$

- As a result, we obtain a set of point matches $(\vec{x}_b, \vec{y}_b)$ that establish the correspondences between I_S and I_T.

- We compute $\vec{u}_{i+1}^{[-1]}(\vec{x})$ using the sampling points $\vec{u}_b(\vec{x}_b) = \vec{y}_b - \vec{x}_b$ to determine T_{i+1}^{-1}.

- We resample I_S: $I_{S,new}(\vec{x}_{new}) = I_{S,i+1}(\vec{x}_{new}) = I_S(T_{i+1}^{-1}(\vec{x}))$.

Stopping criterion:

We continue the iterations until the transformation has become as accurate as desired, meaning that either a certain number of iterations has been performed or a certain threshold value of the similarity measure (see section 2.1.2) is exceeded.

- We stop the iteration if the demands for the quality of the registration are fulfilled. With $T_i = T_{stop} = T_{n-1}$ we have found the desired transformation T.

These components of the algorithm as well as their inputs and outputs are depicted in figure 5.5.

5.2 Calculation of the Transformation in a Multiscale Approach

In section 5.1 the task of finding the transformation iterations T_i and therewith the overall transformation T regarding two images of a certain size is defined. To gain a better accuracy of the transformation estimation, this procedure has to be extended:

The `Baladin` program uses a *multiscale* approach as described in section 2.2.2. Here, we begin the process of calculating the transformation by *subsampling* the source image and the target image to the lowest level $lmax$ of the pyramid. Hence, we deal with images of a small size where the transformation estimation T_{lmax} is computed fast. However, the quality of the resulting transformation needs to be improved. Thus, in the following step we approximate the transformation on level $lmax - 1$ where the images are less subsampled than on level $lmax$. To use the already calculated transformation estimation T_{lmax}, we have to figure a way to apply T_{lmax} to the images of size $level\_size_{lmax-1}$ before starting the transformation estimation of T_{lmax-1} [22]. In the following, the operation of derivating the overall transformation estimation T in the already existing algorithm of `Baladin` is explained:

`Baladin` calculates the transformation estimations using rigid and affine transformation classes. The determined transformation iterations are represented in matrix-vector form (see section 2.2). Likewise, the action of subsampling an image is encoded in a *subsampling* matrix.

When the transformation estimation on a given level l is completed, a transformation matrix for that level is obtained. That transformation matrix is valid only for its respective level l since it is expressed in the level specific image coordinate system. To apply it to the images of the following level $l - 1$, it is first multiplied with the inverse subsampling matrix of the current level and then multiplied with the subsampling matrix of the next level. The procedure is repeated for each level to obtain finally a *result* matrix that contains all transformations of all pyramid levels. In the last step of the algorithm, the reference

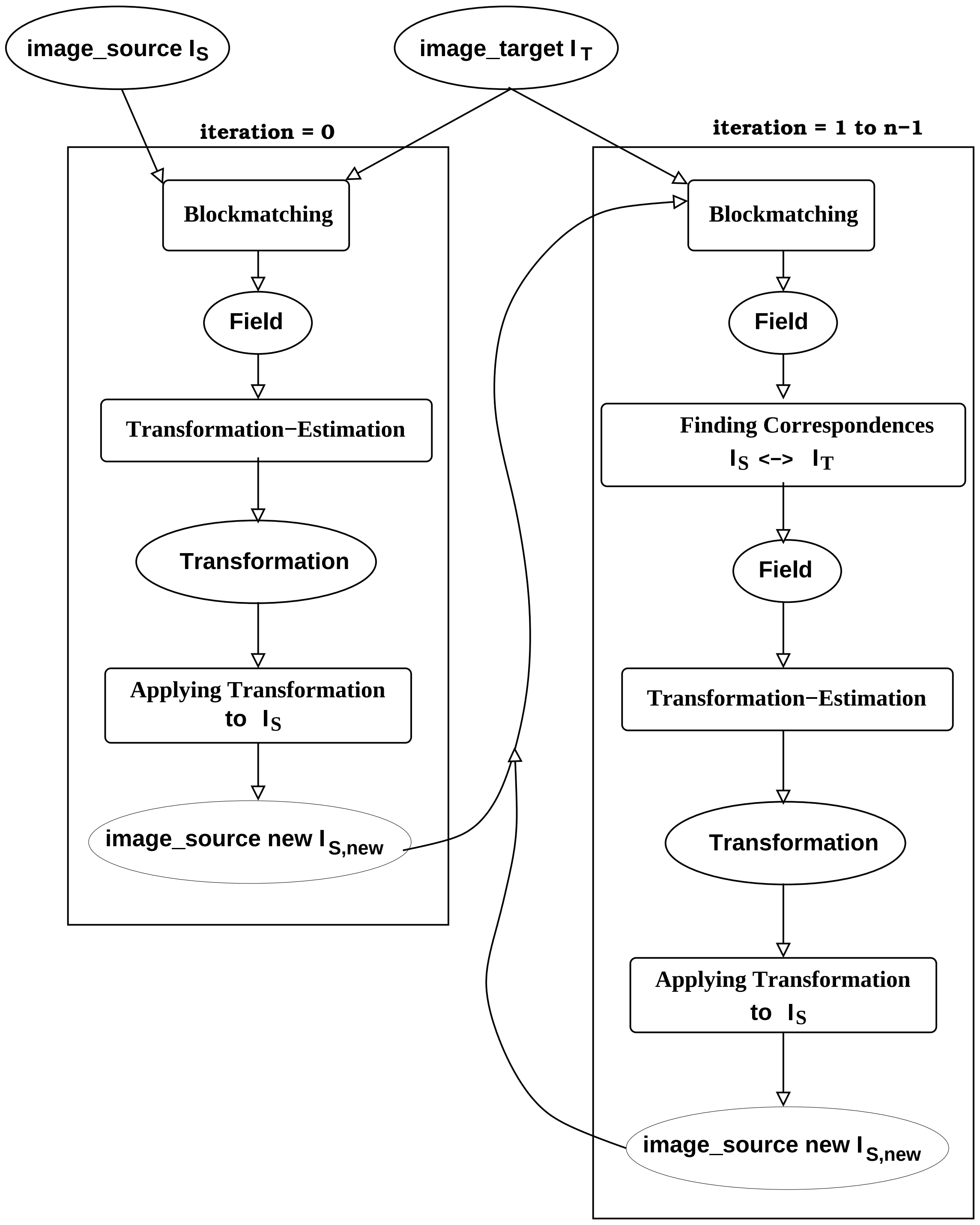

image_source I_S
image_target I_T
iteration = 0
iteration = 1 to n−1
Blockmatching
Field
Transformation−Estimation
Transformation
Applying Transformation to I_S
image_source new $I_{S,new}$
Blockmatching
Field
Finding Correspondences I_S <-> I_T
Field
Transformation−Estimation
Transformation
Applying Transformation to I_S
image_source new $I_{S,new}$

Figure 5.5: *Course of action in the calculation of the B-Spline transformation. On one level of the multiscale pyramid the transformation estimation is iterated as described in section 5.1. The calculating component* Block-Matching *computes a displacement fi eld between two input images, the component* Transformation-Estimation *uses the fi eld as input to compute the transformation. Then, the transformation is applied to image I_S to obtain image $I_{S,new}$. $I_{S,new}$ serves as input for the next iteration. The displacement fi eld containing correspondences between $I_{S,new}$ and I_T as well as the transformation are used in the component* Finding Correspondences*to calculate the new correspondences between I_S and I_T.*

image is deformed by multiplying the result matrix with the voxel position coordinates. (These operations are described in a detailed manner in section B and figure B.2.)

Regarding the B-Spline transformation we have to think of another way to apply a transformation that was calculated on level l to the images of the following level $l - 1$ because there is no direct way to transform a spline function with a (subsampling) matrix.
The data where we start from are the following:

- For level l we know all correspondences $\vec{u}_l$ between the source and target voxels ($v_{S,l}$ and $v_{T,l}$) of the subsampled images.
- All voxels v_l of the subsampled images of level l can be retrieved as voxels v_{l-1} in the subsampled images of level $l - 1$.
- Hence, we know already some correspondences between voxels of the source and target images of level $l - 1$.

Now, we use these correspondences as a set of matches and apply a B-Spline approximation on basis of a least squares estimator to compute a transformation representing T_l on level $l - 1$ [23]. This approach is very convenient concerning the implementation because it resorts to the B-Spline algorithm already implemented (see section 4.1).

CHAPTER 6

Implementation

This chapter deals with the properties and characteristics of the main elements of the implementation. In section 6.1 the integration of the B-Spline transformation as a registration technique in the `Baladin` program is described. In section 6.2 the conjugate gradient algorithm used for solving the Least Squares estimator introduced in section 4.2 is presented in detail. Finally, section 6.3 gives an overview of the complexity of the individual components necessary to estimate the transformation.

6.1 Integration of the B-Spline Transformation in `Baladin`

The B-Spline transformation will be estimated in a pyramidal approach and two nested iteration processes (see figure 6.1). As described in section 2.2.2, we commence the calculation by subsampling the images to the lowest level of the pyramid to estimate a first coarse transformation. On each pyramid level, we iterate the transformation estimation by applying the transformation of the iteration i to the (subsampled) source image I_S to obtain the deformed source images $I_{S,i}$ and using $I_{S,i}$ as input for the next iteration $i+1$. In each iteration, we additionally iterate over the number of control points used for the B-Spline approximation. This multiresolution procedure is explained in section 4.5. Now, these processes must be integrated in the already existing `Baladin` program.

The program `Baladin` was written in C and serves uniquely for registrating two images in a rigid, similitude, or affine manner. To integrate the B-Spline transformation in the `Baladin` program, we have to extensively analyse the code of `Baladin` to determine

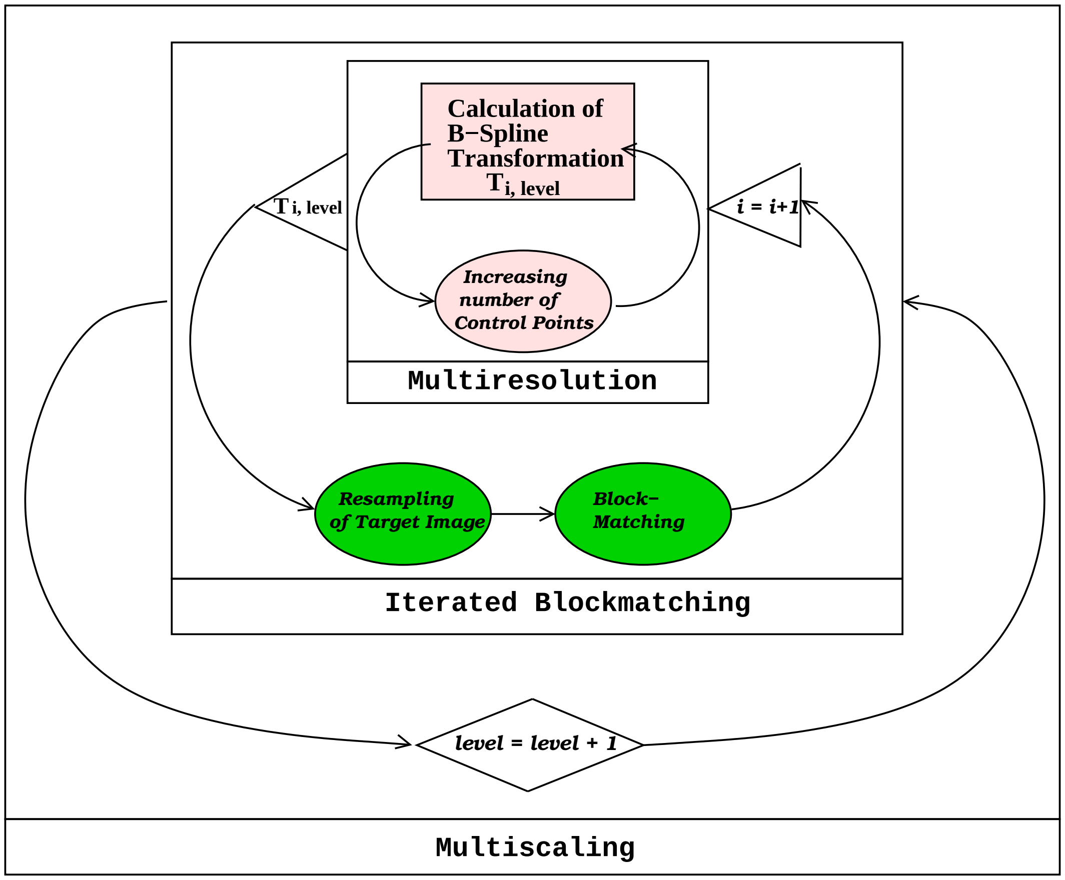

Calculation of B−Spline Transformation $T_{i,\,level}$
Increasing number of Control Points
Multiresolution
$T_{i,\,level}$
$i = i+1$
Resampling of Target Image
Block−Matching
Iterated Blockmatching
level = level + 1
Multiscaling

Figure 6.1: *Diagram of the computation process.*

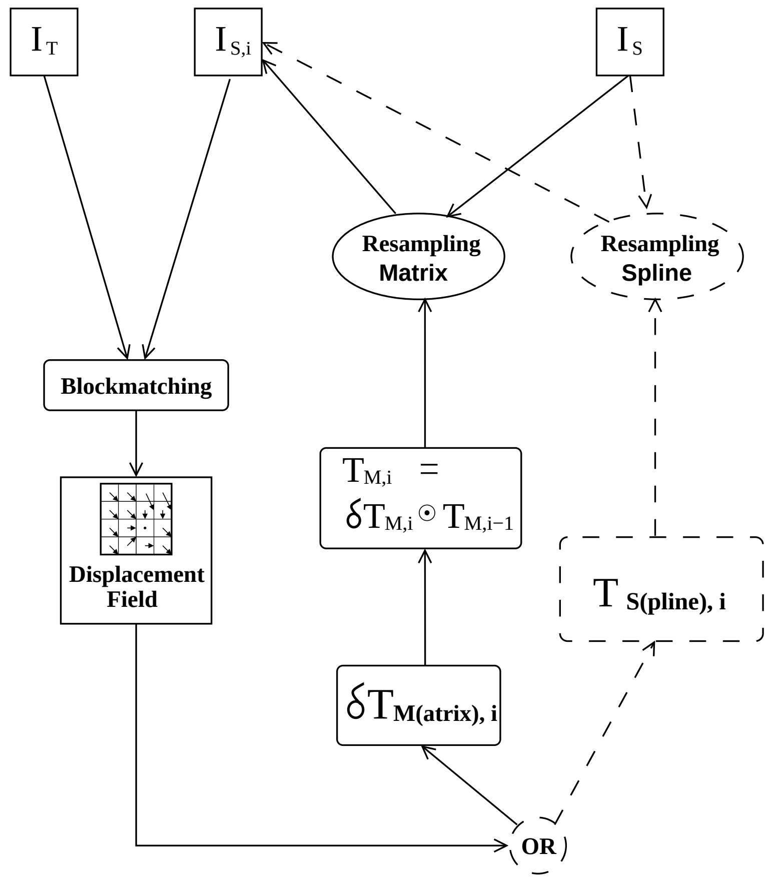


Figure 6.2: *Provisional Structure of `Baladin` and the extension. On a certain level of the multiscale pyramid, the two (subsampled) images I_T and I_S are the inputs for the block-matching algorithm. The displacement fi eld as output of the block-matching algorithm is used as input for the matrix transformation as well as for the B-Spline transformation. Hence, the result of the block-matching algorithm serves as interface to connect the B-Spline transformation. Before setting off the respective next iteration $i + 1$, the image I_S is resampled to image $I_{S,i}$ either using the matrix or the B-Spline transformation.*

the locations in the code that will serve as interfaces. Generally, the program has to be extended as displayed in figure 6.2. Thus, a function that estimates the B-Spline transformation (*`Estimate-Transformation-Spline`*) and a function that resamples the image (*`Resample-Image-Spline`*) have to be integrated. As described in chapter 5, the resampling of an image using the B-Spline transformation in an iterative approach needs to be realized in a different manner than the resampling using a matrix transformation. Taking this into account, the extension has to be implemented as shown in figure 6.3. To obtain the displacement field that serves as input to the B-Spline transformation estimation *two* steps are necessary: At first,we have to apply the block-matching algorithm to find a set of corrrespondences between the images $I_{S,i}$ and I_T. Next, we have to apply the transformation $T_{Spline,i}$ to image $I_{S,i}$ to find the respectively corresponding points in

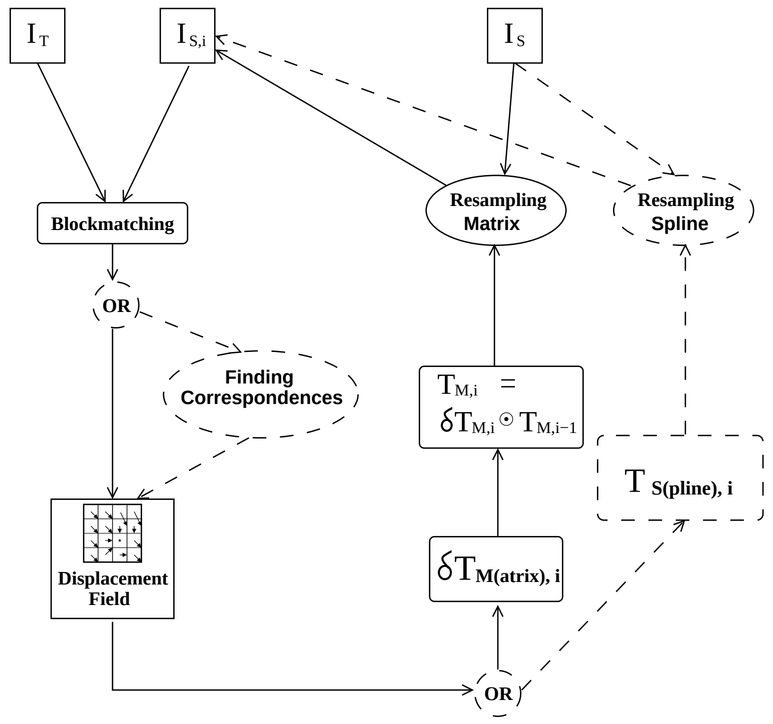


Figure 6.3: *Structure of* `Baladin` *and the extension. The block-matching algorithm calculates the correspondences between I_T and $I_{S,i}$. The component* `Finding Correpondences` *uses the output of the blockmatching algorithm to fi nd a new set of correspondences between I_S and I_T that will be used as input for the spline transformation estimation (for details see chapter 5).*

image I_S. In doing so, we receive a set of correspondences between I_S and I_T that serves as the new displacement field (see section 5.2).

To be easily integrated in the `Baladin` program, the implementation of the B-Spline transformation estimation was written in the programming language C. To integrate the functions concerning the B-Splines that are described above, `Baladin` has to be re-organized to offer the possibility of choosing between different types of transformation classes. For these purposes, a structure named *`TRANSFORM`* is created that contains among other things a component of type *`void*`*. This structure will substitute the present matrix component used to store the transformation. According to the choice of the user, the *`void*`* -component is initialized to be either a matrix structure or a B-Spline structure. All functions of `Baladin` are modified to work on the new structure instead of the matrix. The function calls of the interface are adapted to branch to either the matrix or the B-Spline functions.

Some of the components in the process of implementing of the B-Spline transformation are listed below:

- We create a structure for B-Splines and a file that contains all components needed to create, represent and work with the B-Spline functions.
- We create a c-file named `estimateur_spline.c` that is responsible for all operations to estimate the B-Spline transformation, e.g. the *Least Squares* algorithm.
- We create a c-file named `reech_spline.c` that contains all measures to be taken when resampling an image with a B-Spline transformation.
- To be able to judge the results obtained with the B-Spline transformation, we implement a code that verifies the accuracy of the transformation. Here, the original sampling point values that serve as displacement vectors are compared with the vectors that are created by the transformation for the same voxel (see e.g. section 7.1).

6.2 The Conjugate Gradient Algorithm

In section 4.1 the mathematical details for determining the control points $\vec{p}$ that are needed to represent the B-Spline functions are explained. Thus, we would like to solve the system $B\vec{p} = \vec{u}$. As B is not square but overdetermined we do not know if a solution to that system exists. Thus, instead we will solve the symmetric system below (see also equation (4.5)):

$$\|B\vec{p} - \vec{u}\|_2^2 \overset{!}{\to} \min \tag{6.1}$$

$$\Leftrightarrow \quad B^T B\vec{p} \ = B^T\vec{u}$$

$$\text{with } \vec{u} \in \mathbb{R}^N, \quad \vec{p} \in \mathbb{R}^m, \quad B \in \mathbb{R}^{m\times N}.$$

It is always possible to find a vector $\vec{p}$ that minimizes equation (6.1).

The numbers $N = 600000$ and $m = 8000$ are an example for typical values the parameters could adopt. Hence, B is a big sparse matrix that we do not want to invert or to factorize, therefore, we have to find another idea than calculating $\vec{p}$ directly. A direct approach would involve factoring that will consume a lot of time and memory. We resort to an *iterative* method as most iterative methods are memory-efficient and run fast with sparse matrices.

Since $B^T B$ is symmetric positive definite (as we deal with a matrix of full rank), the equation is a system of normal equations that are typically associated with the least squares problem.

We know $f'(\vec{p}) = B^T B\vec{p} - B^T\vec{u} = 0$ being the derivative of the quadratic system

$$f(\vec{p}) = \frac{1}{2}\vec{p}^T B^T B\vec{p} - \vec{u}^T\vec{p} + c$$

that has to be minimized. As $B^T B$ is positive definite the function f is shaped like a parabolic bowl. Hence, searching the minimum as demanded in equation (6.1) we have to determine the location where the gradient of f equals zero [25].

The most popular approach to solve a non-symmetric linear system in the way described above employs the *conjugate gradient* procedure. It is efficient and will converge theoretically after a certain number of iterations. The number of iterations necessary equals the dimension m of $\vec{p}$. In practice, the accumulated floating point roundoff error causes the residual to gradually loose accuracy. However, this does not pose a problem as in general the results become satisfyingly precise before the procedure has actually undergone m iterations [26]. In the conjugate gradient approach, we look for the solution 'gradient equals zero' by analyzing a $B^T B$-orthogonal set of search directions. Let $M = B^T B$ and $v = B^T\vec{u}$ (see equation (4.6)). The algorithm that is directly applied to the system $Mp = v$ looks like denoted below:

Conjugate Gradient Algorithm:

1. Choose a initial solution $\vec{p}_0$.
2. Compute the first residual $\vec{r}_0 = \vec{v} - M\vec{p}_0$.
3. Set the first search direction $\vec{s}_0 = \vec{r}_0$.
4. For $i = 0$ until convergence do:
   - calculate the length α_i of the distance to go in search direction $\vec{s}_i$,
     $\alpha_i = \frac{\langle \vec{r}_i, \vec{r}_i \rangle}{\langle \vec{s}_i, M\vec{s}_i \rangle}$

- $\vec{p}_{i+1} = \vec{p}_i + \alpha_i \vec{s}_i$
- calculate the residual $\vec{r}_{i+1}$,

  $\vec{r}_{i+1} = \vec{r}_i - \alpha_i M \vec{s}_i$
- $\beta_i = \frac{\langle \vec{r}_{i+1}, \vec{r}_{i+1} \rangle}{\langle \vec{r}_i, \vec{r}_i \rangle}$
- calculate the search direction $\vec{s}$

  $\vec{s}_{i+1} = \vec{r}_{i+1} + \beta_i \vec{s}_i.$

It holds $\vec{r}, \vec{s} \in \mathbb{R}^m$ and $\alpha, \beta \in \mathbb{R}$.

For each iteration we have to compute one matrix-vector multiplication and 5 inner products of size m.

Supposing we want to reduce the norm of the error $e_i = \vec{p}_{Solution} - \vec{p}_i$ by a factor of ε with

$$\|e_i\|_M \;\leq\; \varepsilon \|e_0\|_M$$

and with

$$\|\vec{p}\|_M = \sqrt{\vec{p}^T M \vec{p}}$$

we have to perform the number of i_C iterations. We find

$$\varepsilon \leq 2 \left(\frac{\sqrt{\kappa(M)} - 1}{\sqrt{\kappa(M)} + 1} \right)^{i_C} \tag{6.2}$$

and deduce

$$i_C \leq \frac{1}{2} \sqrt{\kappa(M)} \, \ln(\frac{2}{\varepsilon}). \tag{6.3}$$

where $\kappa(M)$ stands for the *condition number* of matrix M, $\kappa(M) = \|M\| \|M^{-1}\|$ [27]. The foundation on which to decide when to stop the iterating process is explained in section 7.1.

To speed up the process of convergence, we employ a method that improves the iteration situation:

We *precondition* our matrix $M = B^T B$. Applying a preconditioning technique ameliorates the condition number κ of a matrix by down-sizing so that i_C (see equation (6.3))

becomes smaller. To do so, we create a matrix P that has the properties to be symmetric positive definite, to have an appearance similar to M, and to be *more easily invertible* than M so that $\kappa(P^{-1}M) \leq \kappa(M)$ holds. As the matrix $P^{-1}M$ is generally not symmetric nor positive which is required for applying the conjugate gradient algorithm we decompose $P = P_d P_d^T$ (what will always work because P is symmetric and positive definite).
Working with the equation equation (4.6) that was given in section 4.2 we form

$$\begin{aligned} & M\vec{p} && = \vec{v} \\ \Leftrightarrow\quad & P_d^{-1}M(P_d^T)^{-1}P_d^T\vec{p} && = P_d^{-1}\vec{v}. \end{aligned}$$

where $P_d^{-1}M(P_d^{-1})^T$ will be symmetric and positive definite [28]. The new - preconditioned - problem to be solved is now represented by the following equation

$$\begin{aligned} P_d^{-1}M(P_d^{-1})^T\ \vec{\hat{p}} &= P_d^{-1}\vec{v} \\ \text{with}\quad \vec{\hat{p}} &= P_d^T\vec{p} \end{aligned} \tag{6.4}$$

To compute the control point values, we commence by determining $\vec{\hat{p}}$ using the conjugate gradient algorithm and carry on by calculating $\vec{p}$ based on the results.
To simplify the calculation we resort to a diagonal preconditioning in our implementation and define $P_d^{-1} = (P_d^T)^{-1}$ to be a diagonal matrix with the values

$$P_{d,ii}^{-1} = \sqrt{\|M_i\|_2}$$

where M_i denotes the i-th row of matrix M.

6.3 Complexity of the Transformation Estimation

In this section we describe the complexity of the different components of the transformation estimation to determine which element uses how much of the computation time. There are three principal components that need to be analysed (compare [1]):

1. The evaluation of the spline transformation.

2. The assembling of the matrix for the CG-procedure.

Operation	Number θ
*	12
/	6
+ -	6

Table 6.1: *Number of mathematical operations necessary to evaluate the B-Spline basis functions for one point.*

Operation	Number θ	Equivalence
*	$m(3 * 12 + 5))$	m
/	m(3*6)	m
+ -	m(3*6 +1) -1	m

Table 6.2: *Number of mathematical operations necessary to evaluate the B-Spline transformation.*

3. The calculation of the CG-procedure.

1. **The evaluation of the spline transformation:**

   The evaluation of a B-Spline basis functions is implemented using the algorithm of De Casteljau. Here, determining the value of a cubic B-Spline basis function given an input x takes 6 mathematic operations where each one consists of 2 multiplications, 1 division and 1 substraction (see table 6.1) [29].

   To evaluate the value of the B-Spline transformation as presented in equation 4.2 we need to evaluate 3 times a B-Spline basis function, we need to effactuate 2 multiplications for the tensor product and three multiplications for the values of the controlpoints.

   Those operations have to be performed for all m controlpoints. Finally, all m contributions of the control points have to be summed, so there need to be $m - 1$ additions . Resumed, we get the number of operations as displayed in table 6.2.

2. **Assembling of the Matrix:**

Aufwandsbeschreibung der LS-Estimation wie in Declerck, Seite 28-31

hier koennte ich ausserdem die tests mit epsilon im CG-verfahren unterbringen

oder gehoert das in ’results and discussion’?

CHAPTER 7

Results and Discussions

7.1 Performance of the CG-Algorithm

As described in section 6.2 it suffices to perform a certain number of iterations because we reach a satisfactory accuracy of the result before the iteration converges. In this section, we will present the performance of the CG-Algorithm in dependence of the number of iterations executed. The number of iterations executed can either be fixed to a certain figure in the beginning or be dependent of a *break condition*. The break condition establishes a threshold of error that must be undergone by the results of the algorithm.
In the implementation we designate the relative amplitude of the residual $\|r_i\|_2$ as compared to the length of the vector $\vec{v}$ to be the break condition (see equation (7.1)).

$$\begin{aligned} \|r_i\|_2^2 &= \|B^T B\vec{p} - B\vec{u}\|_2^2 \\ &\overset{(4.10)}{=} \|M\vec{p} - \vec{v}\|_2^2, \end{aligned}$$

$$\text{if } \|r_i\|_2^2 \leq \varepsilon\|\vec{v}\|_2^2 \qquad (7.1)$$

$$\text{break!}$$

To determine the performance of the conjugate gradient algorithm in dependence of the value chosen for ε we execute several testing procedures. In the following, the detailed description of the testing procedure and its outcomes comprises four paragraphs; the first one illuminates the input situation where we start, thereupon, the settings for the transformation estimation are displayed in the second paragraph followed by a depiction of the actual execution of the testing in the third paragraph. In the fourth paragraph, the outcomes are analyzed and interpreted.

1. **The input for the CG-testing:**

   The test image volume on hand measures $256 \times 256 \times 256$ voxels. The voxels volume is set to $1 \times 1 \times 1$. To approximate a typical situation when working with the block-matching algorithm, one displacement vector is positioned every 3 voxels. The emerging sparse displacement field that serves as input for the B-spline transformation consists of $N = 85 \times 85 \times 85 = 614125$ displacement vectors.

   To test different types of image situations, we created diverse kinds of the sparse displacement fields that are noted in the following list:

   (a) All sampling point vectors $\vec{u}_b$ have the same values, thus, the field represents a translation.

   (b) The sampling point vectors $\vec{u}_b$ are built by applying a matrix A to their respective position coordinate values $\vec{x}_b$ and then subtracting the deformed vectors from their position vectors:

   $$\vec{u}_b = (A - I)\vec{x}_b. \tag{7.2}$$

   We use one-by-one

   - a matrix A that effects a rotation around the z-axis (representing a rigid transformation),
   - a matrix A that effects rotations around the x-, y-, and z-axis under different angles α, β, γ (representing a rigid transformation),
   - a matrix A that effects an affine transformation

   to deform the position values $\vec{x}_b$.

   (c) The sampling point vectors $\vec{u}_b$ are built by using the position values $\vec{x}_b$ as input for a (three-dimensional) sinusoidal function.

   All testing approaches described above have resulted in satisfying outcomes. To illustrate the insights we gained with these experiments, in the following the testing and the results when using an affine deformed sparse displacement field are

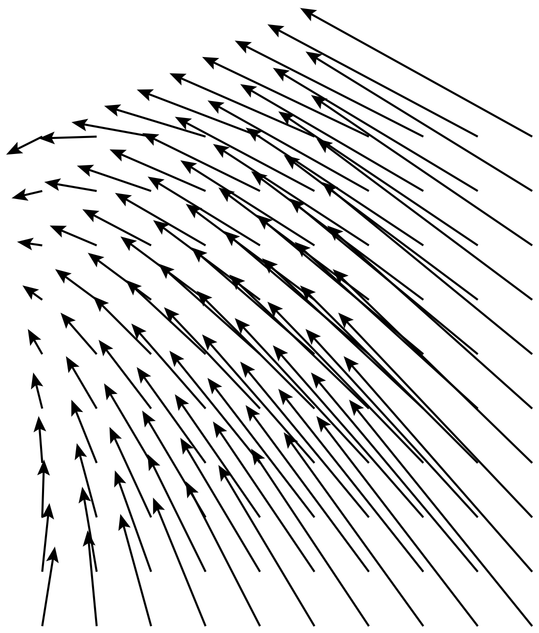


Figure 7.1: *2D Cut of a displacement vector fi eld. It was gained by applying an affi ne matrix to the coordinates of a volume.*

depicted. (The results of the other approaches can be found in Appendix C.) The affine matrix A_{aff} that warps the position coordinate values $\vec{x}$ into a sparse displacement field (see equation (7.2)) looks like this:

$$A_{aff} = \begin{pmatrix} 0.6 & -0.3 & 0.4 \\ 0.1 & 0.8 & 0.3 \\ -0.4 & 0.2 & 1 \end{pmatrix}.$$

HIER KOMMT NOCH EIN BEWEIS HIN, WARUM DIE MATRIX AFFIN IST

Figure 7.1 shows as an example the resulting displacement vectors of one layer when A_{aff} is applied to a $30 \times 30 \times 30$ volume that is assigned one displacement vector each three voxels.

2. **The parameters of the B-Spline transformation:**

   For each dimension, a quantity of 20 uniformly distributed control points is chosen so that $m = 20 \times 20 \times 20 = 8000$, that is, one control point appears approximately every 4 sampling points.

   The sought-after values for the control points $\vec{p}$ that minimize the equation $||M\vec{p} - \vec{v}|| \overset{!}{=} \min$ are calculated with the conjugate gradient procedure. The factor ε stands for the ratio of the system residual $r_i = Ax_i - b = M\vec{p}_i - \vec{v}$ (see 'conjugate gradient algorithm' in section 6.2) and $\vec{v}$. Therefore, ε is the parameter that controls the number of iterations and that represents the break condition of the conjugate gradient procedure. The loop of iterations is stopped as soon as $\|M\vec{p}_i - \vec{v}\| = \|r_i\| \leq \varepsilon\|\vec{v}\|$ (see (7.1)). The bigger the number of iterations gets the more accurate the final result for $\vec{p}$ becomes. Yet simultaneously, the time consumed by the calculation grows. A reasonable trade-off between the duration of the calculation and the accuracy of the result must be found.

To determine the accuracy of the obtained $\vec{p}$, we apply the B-spline transformation $T_{BSpline}$ that is built on them to the 614125 sampling point positions in the image volume. The vectors calculated at the sampling point positions by $T_{BSpline}$ should equal the original sampling point vectors that were derived from the block-matching algorithm. Now, their respective differences are summed up and standardized to give an error measure θ for the accuracy of the transformation

$$\theta = \frac{1}{N}\sum_{i=1}^{N}(T_{BSpline}(\vec{x}_i) - \vec{x}_i - \vec{u}_i)^2. \tag{7.3}$$

$T_{BSpline}$:	*B-Spline transformation that calculates the displacement vectors for each voxel i*
$\vec{x}_i$:	*position of voxel i*
$\vec{u}_i$:	*sampling point/ displacement vector of voxel i*
N :	*number of sampling points*

3. **The testing procedure:**

The value of parameter ε in the conjugate gradient function of the program `testCodeSplineTransfo` is varied from $3 \cdot 10^{-3}$ to $3 \cdot 10^{-6}$. Each time the program `testCodeSplineTransfo` is executed, the accuracy of the result and the computing time are determined. Here, the accumulated and standardized differences of original and calculated displacement vectors serves as a measure for the accuracy. The computing time depends on the number of iterations necessary to obtain a result that falls below the threshold defined by ε in the conjugate gradient procedure (see section 6.2).

The coherences *epsilon* $\leftrightarrow$ *accuracy* are represented in figures 7.2 and 7.3 to show how the accuracy of the result depends on the value of the break condition. Figures 7.4 and 7.5 display how the choice of ε influences the computation time. The respective correlation *time* $\leftrightarrow$ *accuracy* is presented in figures 7.6 and 7.7.

4. **Conclusion:**

   On the whole, the outcomes of the testing are highly satisfying because a good accuracy is achieved in a well acceptable time as can be clearly seen in figures 7.6 and 7.7.

   As explained above, we want to determine a reasonable trade-off between the computational time consumed and the precision of the result. Analyzing figure 7.4 we observe that the curve can be represented by two evens with different gradients that intersect approximately at coordinate $(35; 48s)$. Here, we reckon to find the optimal compromise regarding CPU-time and accuracy. Using the next sampling value at coordinate $(30; 54s)$ we detect the associated error value as deducted from figure 7.3 being $22 \cdot 10^{-6}$. An error of that order of magnitude fulfills our request concerning the accuracy.

   As we work with float values that have an accuracy of $1 \cdot 10^{-6}$ we do not demand for a greater preciseness. Referring to the literature (or to the unpublished knowledge of persons with a lot of experience regarding image analysis) we decide that an accuracy of $2 \cdot 10^{-4}$ suffices for our application that registrates images volumes on voxel basis.

   Augmenting the error sum tolerated by factor 10, that is an error of $200 \cdot 10^{-6}$, gives a CPU-time value of 37s (see figure 7.4). This is an improvement in time of only 30% compared to 54s at an error sum of $22 \cdot 10^{-6}$. Seeing this, we state that we do not gain a computational time reduction that is worth the loss of accuracy. For an illustration of this context see table 7.1.

   On the other hand, observing our results we can state that the implemented algorithm offers the possibility of improving the accuracy of the transformation without a problematic loss in time. This might be interesting for other applications.

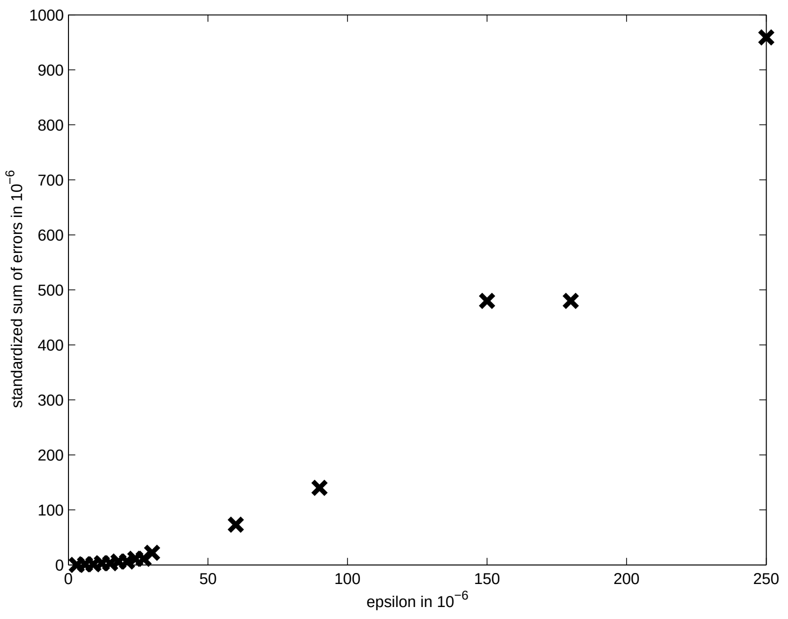


Figure 7.2: *Coherence epsilon ↔ differences. The differences between calculated and original sampling point vectors in dependence of the break condition epsilon are displayed as 'standardized sum of errors'θ (see equation (7.4)).*

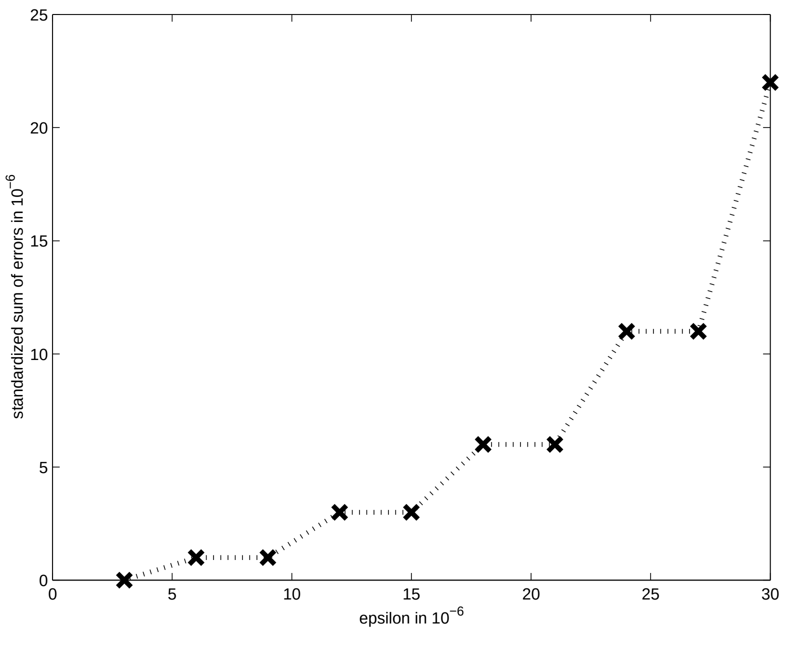


Figure 7.3: *Zoomed in coherence epsilon ↔ differences. The differences between calculated and original sampling point vectors in dependence of the break condition epsilon are displayed as 'standardized sum of errors' θ (see equation (7.4)).*

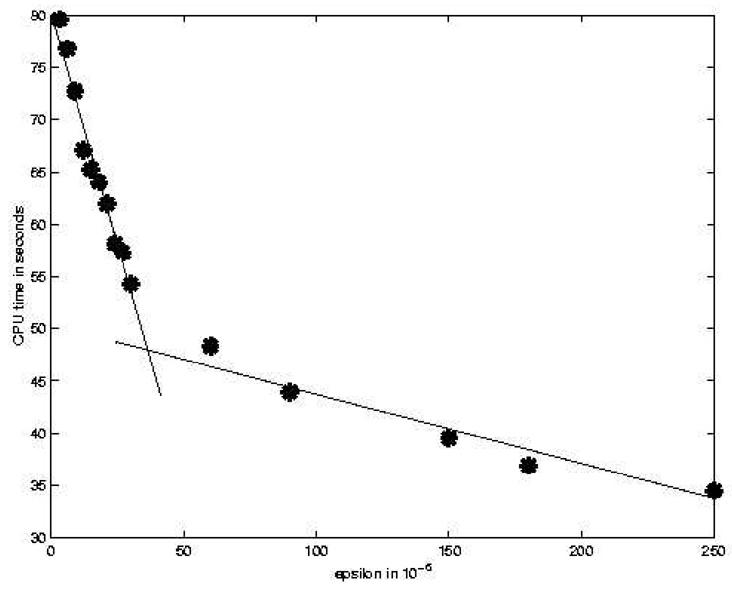


Figure 7.4: *Coherence of the break condition $\varepsilon \leftrightarrow$ time. The time necessary to calculate the CG-procedure in dependence of the break condition ε is displayed.*

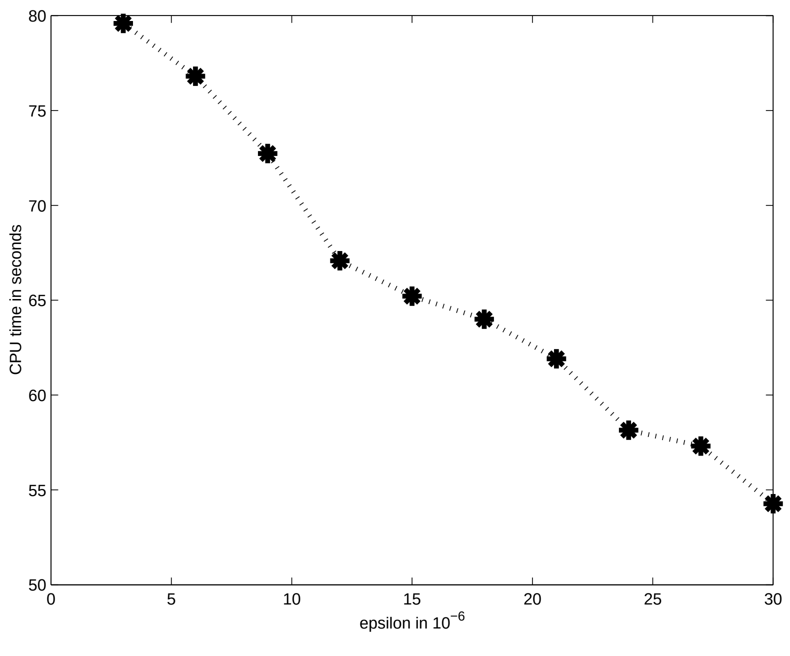


Figure 7.5: *Zoomed in coherence of the break condition $\varepsilon \leftrightarrow$ time. The time necessary to calculate CG-procedure in dependence of the break condition ε is displayed.*

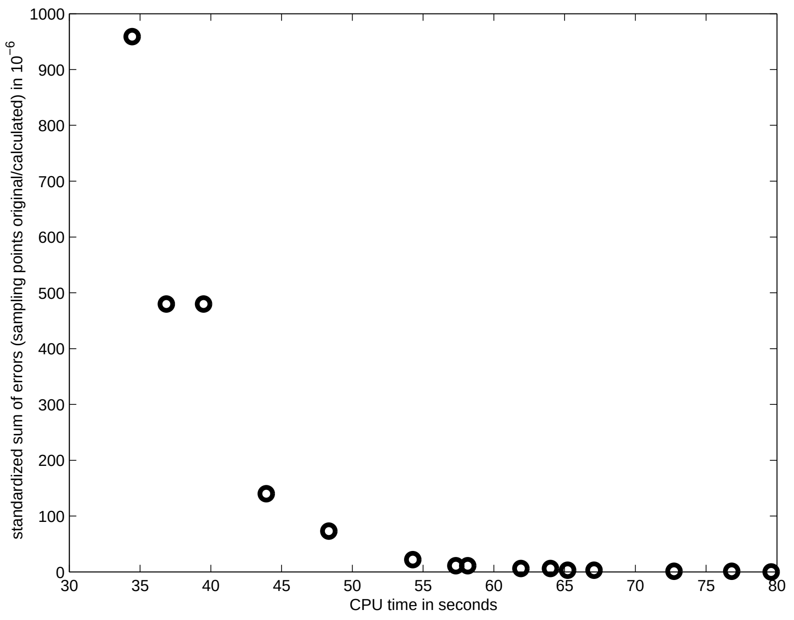


Figure 7.6: *Coherence time ↔ differences. The correspondence of calculation time and standardized sum of errors θ is displayed.*

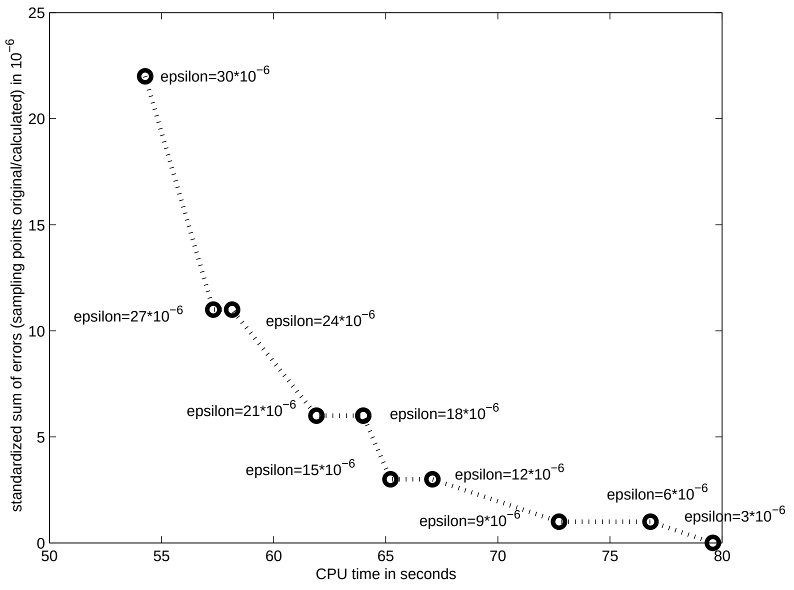


Figure 7.7: *Zoomed in coherence time ↔ differences. The correspondence of calculation time and standardized sum of errors θ is displayed.*

ε	Error θ	CPU-time
$250 \cdot 10^{-6}$	$1 \cdot 10^{-3}$	37s
$100 \cdot 10^{-6}$	$2 \cdot 10^{-4}$	42s
$30 \cdot 10^{-6}$	$2 \cdot 10^{-5}$	54s
$10 \cdot 10^{-6}$	$2 \cdot 10^{-6}$	70s

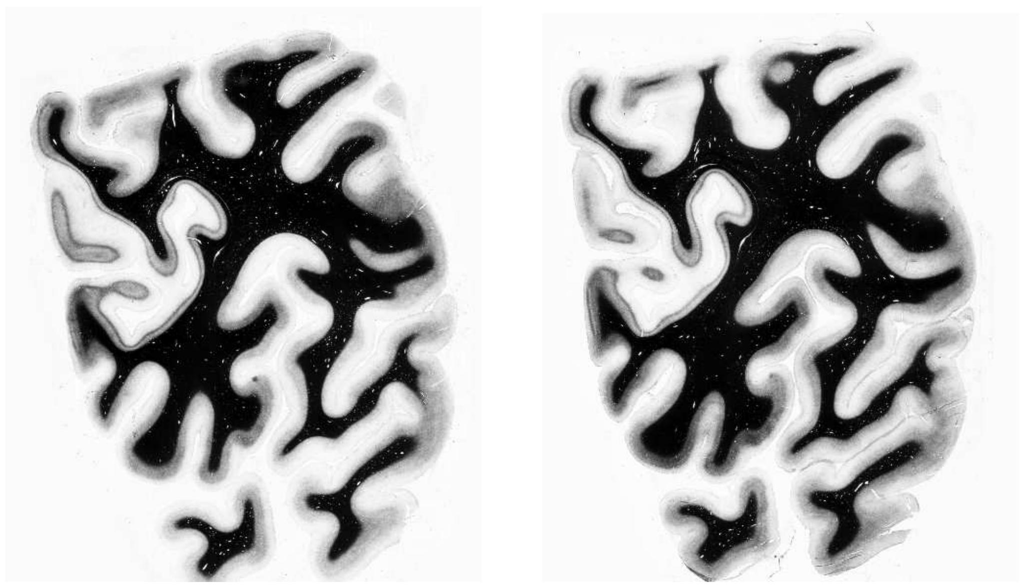
Table 7.1: *Relations between break condition ε, standardized sum of errors θ, and CPU-time. The 3D testing fi eld represented an affi ne transformation. The transformation was calculated on a Pentium III, $1GH$, $1GB$ RAM.*

7.2 Performance of the B-Spline Transformation

In this section, we present first results of the B-Spline transformation. As will be outlined, the basic transformation estimation must be extended by a regularization term to deal with unfavourable effects caused by noise and homogeneous regions contained in the images. The medical images we work ON are histological slices of the brain, VON WO? In the following examples, the source image I_S measures 2936×3811 pixels, the target image I_T consists of 2936×3698 pixels. Their voxeldimensions are $1 \times 1 \times 1$. The images are depicted in figure 7.8.

We apply the B-Spline transformation as described in section 4.2 and chapter 5 with a weigthed Least Squares approach to registrate the two images. The resulting displacement vectors for the source image can be seen in figure 7.9. They lead to the deformed source image I_S that can be seen in figure 7.10.

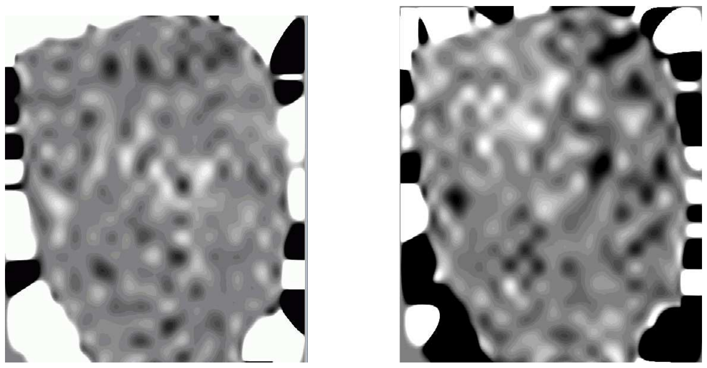
The incorrectly interfering patterns at the borders of the image are clearly visible. Evidently, performing further iterations using the results of the first one leads to an augmented error effect ending in a total destruction of the original image. The error effects are due to a combination of homogeneous regions in the image covered with noise. When we regard the distribution of sampling points (see figure 7.11) we come to the conclusion that the error effects appear only in the parts of the image where are none. As explained in section 4.3, we use a *weighted* Least Squares approach where sampling points coming from regions with high significance are assigned a greater weight than sampling points coming from regions with low significance. Big homogeneous regions have zero significance, thus, the sampling points are eliminated from the input for the B-Spline approximation.



(a) *The source image I_S is a histological slice of the brain that measures* 2936×3811 *pixels.*

(b) *The source image I_S is a histological slice of the brain that measures* 2936×3811 *pixels.*

Figure 7.8: *The source image I_S and the target image I_T that are used in several experiments. The grey scale varies from* 0 *to* 237.



(a) Value of displacement vector in x-direction

(b) Value of displacement vector in y-direction.

Figure 7.9: *Displacement values for a registration of the images shown in 7.8. We used 25 control points per direction and performed one iteration on level 4 of the* unregularized *B-Spline transformation estimation.*

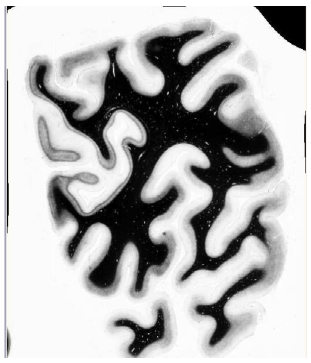


Figure 7.10: *Deformed source image $I_{S,new}$ after one iteration on level 4 of the* unregularized *B-Spline transformation estimation.*

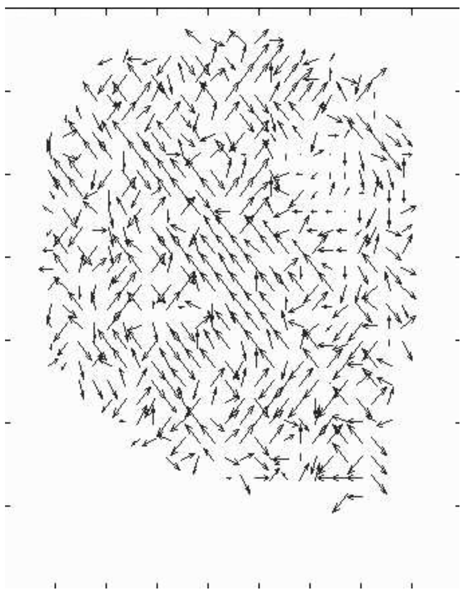


Figure 7.11: *Distribution of sampling points when registering the images of fi gure 7.8. In the homogeneous regions of the images, no sampling points are found.*

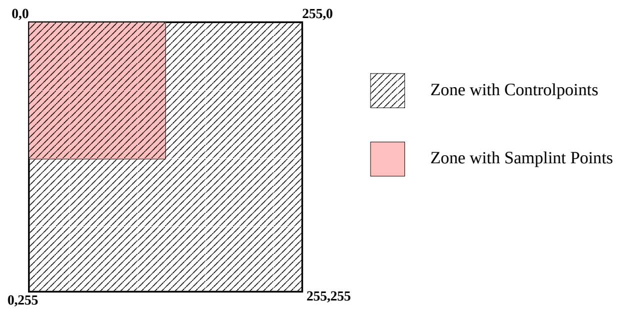


Figure 7.12: *Example of an output of the block-matching algorithm when the images contain great homogeneous regions. Here, all sampling points are situated in the upper left quarter of the image. The control points are distributed over the whole of the image.*

7.2.1 Effects of Noise and Homogeneous Regions

The B-Spline functions represent the approximation of the transformation as described in section 4.1 and equation (4.2):

$$\vec{u}_{BSplines}(x,y,z) = \sum_{l=0}^{3}\sum_{m=0}^{3}\sum_{n=0}^{3} B_l^x(\alpha)B_m^y(\beta)B_n^z(\gamma)P_{i+l,j+m,k+n},$$

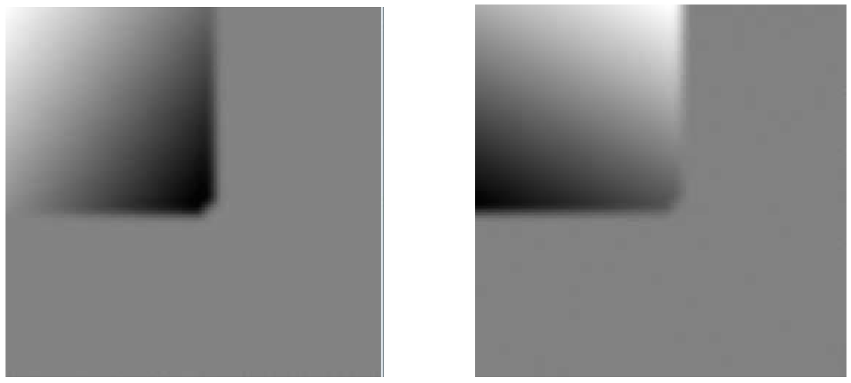
$$P_{i,j,k},\ \vec{u}_{BSplines}(x,y,z)\ \in\ \mathbb{R}^3.$$

The optimal control points $P_{i,j,k}$ are found by minimizing the Least Squares estimator

$$C = \sum_{s=0}^{N-1} \|\vec{u}_b(\vec{x}_b) - \sum_{l=0}^{3}\sum_{m=0}^{3}\sum_{n=0}^{3} B_l^x(\alpha)B_m^y(\beta)B_n^z(\gamma)P_{i+l,j+m,k+n})\|^2.$$

Assuming that the control points are uniformly distributed across the image and the sampling points are not, we are faced with the problem that the values of several control points cannot be approximated due to lack of data in the respective region. To analyse the resulting effect, we perform testing procedures by registrating two images that simulate great homogeneous regions. The respective displacement field put out by the block-matching looks like the scheme portrayed by figure 7.12.

All sampling points are comprised in one region that takes the upper left querter of the image. The other three quarters contain no sampling points. The control points used to approximate the B-Spline transformation are uniformly distributed all oder the image. The simulated sampling points have values in the range of 10^2. The image is of size 256×256. We simulate 2 different situations:



(a) *Value of displacement vector in x-direction.* (b) *Value of displacement vector in y-direction.*

Figure 7.13: *Displacement vectors for all pixels in an image measuring* 256×256 *pixels where no noise was added. All sampling points were situated in the upper left quarter of the image. The control points were uniformly distributed. The type of grey covering most of the image represents the value zero. The scale of grey is of magnitude* $[-10^2, 10^2]$.

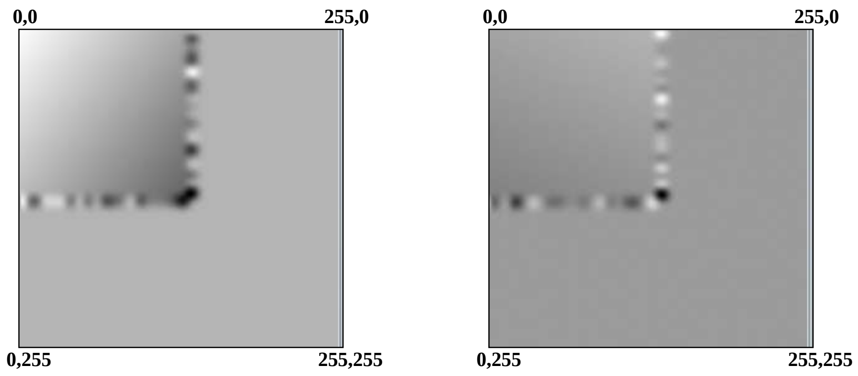
1. The images contain no noise effect.

2. The images contain noise effects. The noise used is random and has a mean of zero. The bounds of the noise are varied; for the following image examples we chose the highest amplitude to be in the magnitude of the length of the sampling points.

Now, we approximate the B-Spline transformation using the displacement field shown in figure 7.12. Subsequently, we apply the B-Spline transformation to the source image. Then, the resulting displacement vectors of the image pixels are displayed.

1. In case of no noise, we obtain the images shown in figure 7.13. All control points were initialised to zero, therefore, the displacements in the region where no sampling points were found equal likewise zero. In the upper left quarter of the image, the displacement vectors are calculated correctly to values other than zero.

2. In case of noise added to the images, the results look like demonstrated in figure 7.14. Like in the example without noise, the displacement vectors in the left upper quarter of the image were approximated correctly by the B-Spline transformation. However, at the border between the region that contains sampling points and the region that contains none, the displacement vectors calculated do not at all represent the desired transformation. The noise effectuates an impulse at the border that



(a) *Value of displacement vector in x-direction.*

(b) *Value of displacement vector in y-direction.*

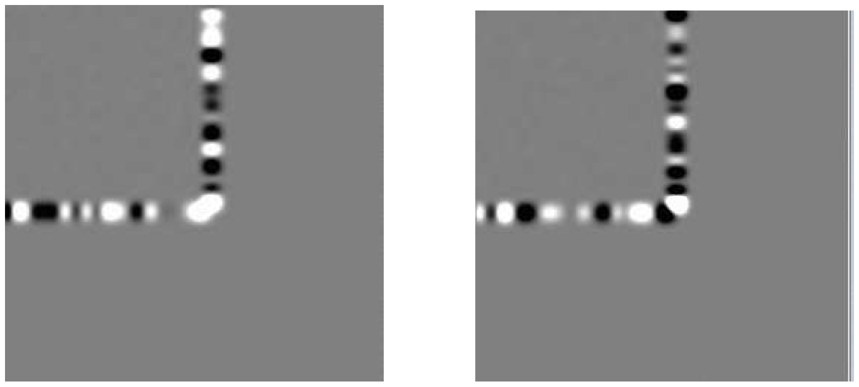
Figure 7.14: *Displacement vectors for all pixels in an image measuring* 256×256 *pixels where a zero-mean random noise was added. All sampling points were situated in the upper left quarter of the image. The control points were uniformly distributed. The type of grey covering most of the image represents the value zero. The scale of grey is of magnitude* $[-10^3, 10^3]$.

cannot be compensated by the next row of control points as those do not take part in the estimation. Hence, the impulse given to the Least Squares approach leads to unpredictable results and therefore to invalid displacement vectors at the border.

To judge the deviation of the transformation estimated on images with noise or on images without nose, we compute the differences between the resoective displacement values. In figure 7.15 the difference images are displayed. Evidently, the noise does only effect the contrlo points at the borders between inhomogeneous and homogeneous regions. The transformation is still calculoated correctly for the regions that contain sampling points. Considering the results shown above, we have to introduce a regularization term as described in section 4.4. Therewith, we add a constraint to the output of the transformation estimation.[1] We want to obtain a function that does not contain discontinuities to forestall effects like seen in figure 7.14. To demonstrate the influence of the regularization term on the estimation, we now apply the *regularized* B-Spline transformation to the images. The resulting displacement vectors are displayed in figure 7.16 for the no noise example and in figure 7.17 for the noisy example.

As can be seen, the unfavourable effects of the combination of noise and homogeneous regions in an image are heavily reduced. Here, the Least Squares estimator computes non-

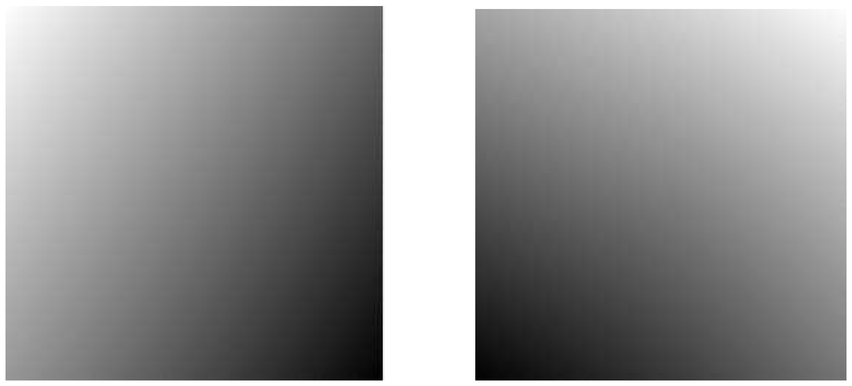
[1]The implementation of the cost function is realized using the Gauss-Legendre technique to compute the integrals.



(a) *Differences of the x values in images 7.13 and 7.14. The values outside the region containing sampling points are all zero (grey scale: $min = -200, max = 200$).*

(b) *Differences of the x values in images 7.13 and 7.14. The values outside the region containing sampling points are all zero (grey scale: $min = -200, max = 200$).*

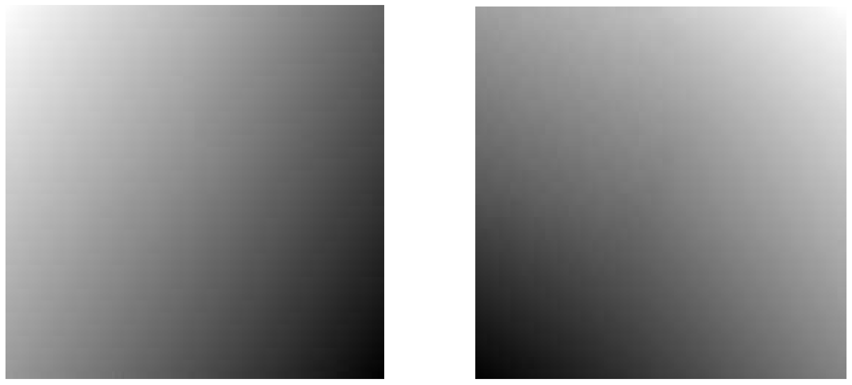
Figure 7.15: *Difference values of images 7.13 and 7.14. The differences outside the region containing sampling points are all zero. The differences inside the region vary between -4 and 4.*



(a) *Value of displacement vector in x-direction.*

(b) *Value of displacement vector in y-direction.*

Figure 7.16: *Displacement vectors for all pixels in an image measuring 256×256 pixels where no noise was added. All sampling points were situated in the upper left quarter of the image. The control points were uniformly distributed. The B-Spline transformation estimation was regularized.*

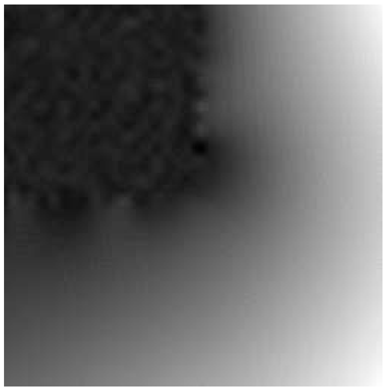


(a) *Value of displacement vector in x-direction.*

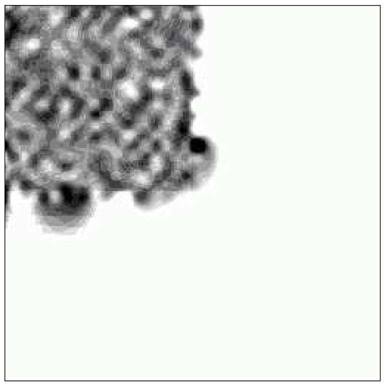
(b) *Value of displacement vector in y-direction.*

Figure 7.17: *Displacement vectors for all pixels in an image measuring* 256×256 *pixels where random zero-mean noise was added. All sampling points were situated in the upper left quarter of the image. The control points were uniformly distributed. The B-Spline transformation estimation was regularized.*

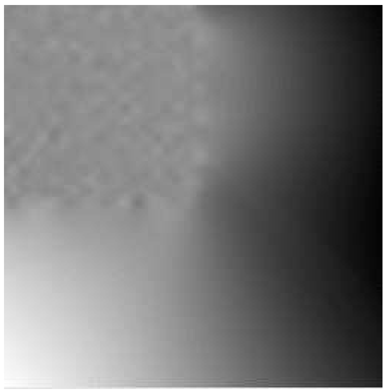
zero values for the control points that lie outside the sampling point region. This results in a slow decay of displacement values beginning at the border of the region containing sampling points and leading toward the edges of the image. The values of displacement in the region with sampling points are still calculated correctly. The difference of the regularized B-Spline approximation for the image example with noise and the image example without noise is displayed in figure 7.18. Looking at figure 7.18 we see that the displacement differences outside the sampling point region are quite big (in the same magnitude as the displacement values), hence, the characteristics of the decay change depending on if the regularization term is applied to images with noise or without noise. Though, the region containing sampling points shows very little difference, therefore, the smoothing effect of the regularization term is satisfying.



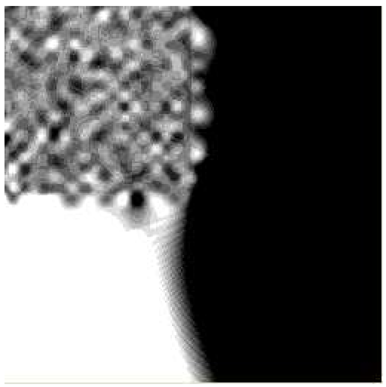
(a) *Differences of the x values in images 7.16 and 7.17, (grey scale:* $min = -10$*,* $max = 75$*).*



(b) *Differences of the x values in images 7.16 and 7.17, (grey scale:* $min = -4, max = 4$*).*



(c) *Differences of the y values in images 7.16 and 7.17, (grey scale:* $min = -63$*,* $max = 50$*).*



(d) *Differences of the y values in images 7.16 and 7.17, (grey scale:* $min = -4, max = 4$*).*

Figure 7.18: *Difference values of images 7.16 and 7.17. The left images show that the differences outside the sampling point region are big whereas the differences inside the sampling point region are quite small. The right images focus on the values in the sampling point region.*

CHAPTER 8

Summary

CHAPTER A

Transformation Parameters

Rigid transformation:

The matrix R of a three-dimensional rigid transformations $T(\vec{x}) = R\vec{x} + t$ consists of three rotation matrices:

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\phi_x) & -sin(\phi_x) \\ 0 & sin(\phi_x) & cos(\phi_x) \end{pmatrix}$$

$$R_y = \begin{pmatrix} cos(\phi_y) & 0 & sin(\phi_y) \\ 0 & 1 & 0 \\ -sin(\phi_y) & 0 & cos(\phi_y) \end{pmatrix}$$

$$R_z = \begin{pmatrix} cos(\phi_z) & -sin(\phi_z) & 0 \\ sin(\phi_z) & cos(\phi_z) & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Affine transformation:

The matrices A_{sc} and A_{sh} of $T(\vec{x}) = A_{sc}A_{sh}R\vec{x} + t$ provide the scaling and shearing parameters for the affine transformation:

$$A_{sc} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{pmatrix}$$

with S_x, S_y, S_z as scaling coefficients and

$$A_{sh} = \begin{pmatrix} 1 & 0 & 0 \\ \alpha & 1 & 0 \\ \beta & \gamma & 1 \end{pmatrix}$$

with α, β, γ as shearing coefficients.

CHAPTER B

Structure of `Baladin` Implementation

The basic structure of the `Baladin` implementation is displayed in figure B.1. The four main c-files are *baladin.c*, *py_image.c*, *estimateur.c*, and *bal_image.c*. As can be seen, the pyramidal approach and the block-matching iteration are realized in *py_image.c* that calls fnctions in *estimateur.c* to estimate the transformation. In *bal_image.c* the resampling (french: “reéchantillonnage”) of the images is executed.

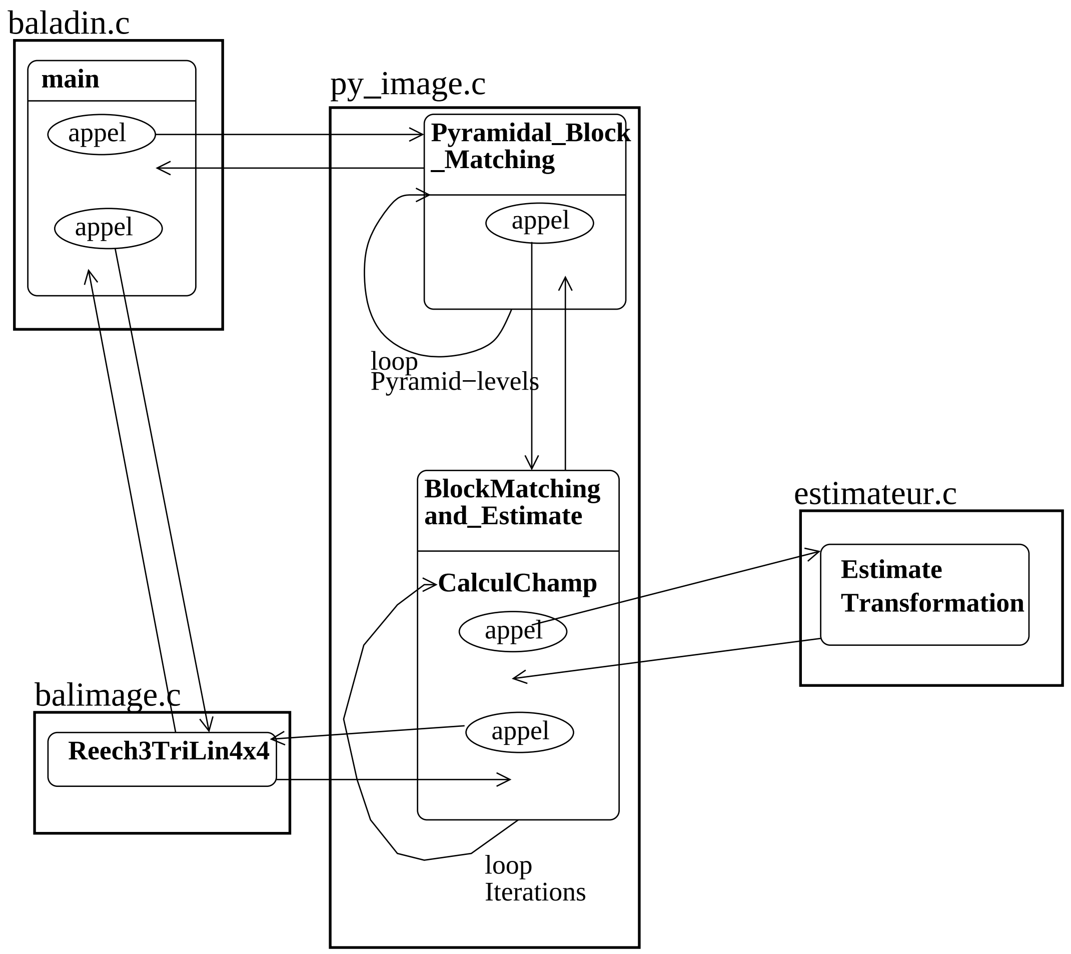


Figure B.1: *Function calls in the baladin implementation using affi ne matrices for the transformation computation.*

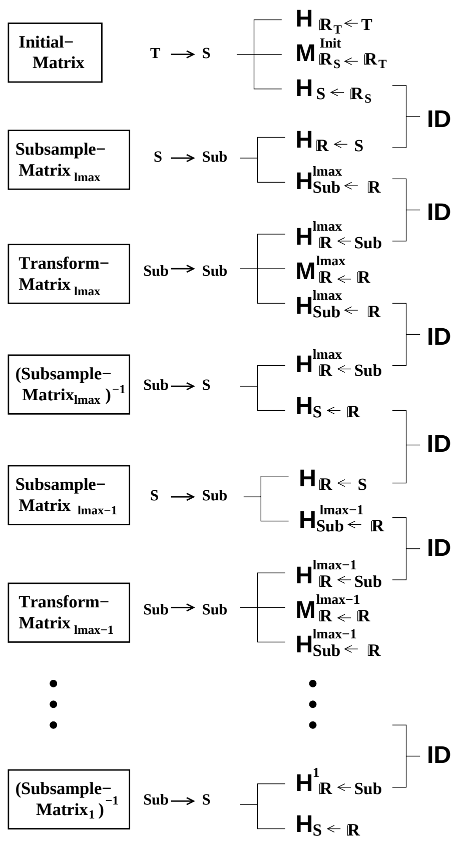
Implementation of the *linear* transformation in a multiscale approach:

In the following, the estimation of a transformation matrix on level l and its application to an image on level $l-1$ is explained.

First, the images are subsampled to their respective size on level l. The action of subsampling an image is encoded in a matrix that is called *Subsample-Matrix*. The *Subsample-Matrix* transforms the image from a voxel into a real coordinate system before actually subsampling it.

When the transformation estimation on a given level l is completed, a transformation matrix *Transform-Matrix*$_l$ is obtained. That transformation matrix is valid only for its respective level l since it is expressed in the level specific image coordinate system. To apply it to the images of the following level $l-1$, it must be first multiplied with the inverse subsampling matrix *(Subsample-matrix)*$_l{}^{-1}$ of the current level and then multiplied with the subsampling matrix *Subsample-Matrix*$_{l-1}$ of the next level. (As these multiplications take place in the real space, no data will be lost due to the subsampling operations as can be seen in figure B.2.) The procedure is repeated for each level.

The matrix *Result-Matrix* is the result of multiplying all matrices used. It contains all information of the transformation estimations of the different pyramid levels. In the last step of the algorithm, the reference image is finally deformed by multiplying the *Result-Matrix* with the voxel position coordinates.



Initial− Matrix
T ⟶ S
$H_{\mathbb{R}_T \leftarrow T}$
$M^{Init}_{\mathbb{R}_S \leftarrow \mathbb{R}_T}$
$H_{S \leftarrow \mathbb{R}_S}$
ID
Subsample− Matrix lmax
S ⟶ Sub
$H_{\mathbb{R} \leftarrow S}$
$H^{lmax}_{Sub \leftarrow \mathbb{R}}$
ID
Transform− Matrix lmax
Sub ⟶ Sub
$H^{lmax}_{\mathbb{R} \leftarrow Sub}$
$M^{lmax}_{\mathbb{R} \leftarrow \mathbb{R}}$
$H^{lmax}_{Sub \leftarrow \mathbb{R}}$
ID
(Subsample− Matrix lmax)−1
Sub ⟶ S
$H^{lmax}_{\mathbb{R} \leftarrow Sub}$
$H_{S \leftarrow \mathbb{R}}$
ID
Subsample− Matrix lmax−1
S ⟶ Sub
$H_{\mathbb{R} \leftarrow S}$
$H^{lmax-1}_{Sub \leftarrow \mathbb{R}}$
ID
Transform− Matrix lmax−1
Sub ⟶ Sub
$H^{lmax-1}_{\mathbb{R} \leftarrow Sub}$
$M^{lmax-1}_{\mathbb{R} \leftarrow \mathbb{R}}$
$H^{lmax-1}_{Sub \leftarrow \mathbb{R}}$
ID
(Subsample− Matrix1)−1
Sub ⟶ S
$H^{1}_{\mathbb{R} \leftarrow Sub}$
$H_{S \leftarrow \mathbb{R}}$

$$\boxed{M_{Result} = H_{S \leftarrow \mathbb{R}} \cdot M^{1}_{\mathbb{R} \leftarrow \mathbb{R}} \cdot \ \dots \ \cdot M^{lmax-1}_{\mathbb{R} \leftarrow \mathbb{R}} \cdot M^{lmax}_{\mathbb{R} \leftarrow \mathbb{R}} \cdot M^{Init}_{\mathbb{R}_S \leftarrow \mathbb{R}_T} \cdot H_{\mathbb{R}_T \leftarrow T}}$$

Figure B.2: *Composition of the overall transformation matrix M_{Result}in the affi ne registration using a pyramidal approach. The fi rst transformation $Transform - Matrix_{lmax}$ is calculated on the lowest level $lmax$ of the pyramid. Each $Transform - Matrix$ consists of a matrix $H_{\mathbb{R} \leftarrow Sub}$ that transforms the subsampled image from the voxel to the real coordinate system, a matrix $M_{\mathbb{R} \leftarrow \mathbb{R}}$ that contains the transformation estimation in the real coordinate system and a matrix $H_{Sub \leftarrow \mathbb{R}}$ that transforms the real into the voxel coordinate system. Each $Subsample - Matrix$ consists of a matrix $H_{S \leftarrow \mathbb{R}}$ that transforms the original source image into the real coordinate system and a matrix $H_{Sub \leftarrow \mathbb{R}}$ that transforms the image of the real coordinate system into a subsampled image in the voxel coordinate system. As can be seen, those matrices fuses to identity matrices so that fi nally all transformation matrices of the different levels are multiplied directly.*

CHAPTER C

Results of the CG-Testing

The conjugate gradient procedure is one of the main components of the transformation estimation presented in this thesis. Here, the results concerning the testing of the conjugate gradient procedure are depicted. We performed different tests by varying the input field that served as hand-made output of the block-matching algorithm (detailed description see section 7.1).

- We create a field that represents a translation. All displacement vectors $\vec{u}_b(\vec{x}_b)$ have the same value:

$$\vec{u}_b(\vec{x}_b) = \begin{pmatrix} 40 \\ 25 \\ 18 \end{pmatrix} \quad \forall\, b \in [1, ..., N]$$

  The results are resumed in table C.1 As can be seen, the results are satifyingly correct and very accurate.

ε	Error $\theta^{1/2}$	CPU-time
$250 \cdot 10^{-6}$	$12 \cdot 10^{-3}$	31s
$100 \cdot 10^{-6}$	$8 \cdot 10^{-3}$	36s
$30 \cdot 10^{-6}$	$6 \cdot 10^{-3}$	40s
$10 \cdot 10^{-6}$	$3 \cdot 10^{-4}$	96s

Table C.1: *Relations between break condition ε, errors of result θ^2, and CPU-time. The 3D testing fi eld represented a translation. The transformation was calculated on a Pentium III, $1GH$, $1GB$ RAM.*

ε	Error $\theta^{1/2}$	CPU-time
$250 \cdot 10^{-6}$	6.14	197s
$100 \cdot 10^{-6}$	6.14	243s
$30 \cdot 10^{-6}$	6.14	243s
$10 \cdot 10^{-6}$	6.14	341s

Table C.2: *Relations between break condition ε, errors of result θ^2, and CPU-time. The 3D testing fi eld represented a sinusoidal transformation. The transformation was calculated on a Pentium III, $1GH$, $1GB$ RAM.*

- We create a sinusoidal field. The displacement vectors $\vec{u}_b(\vec{x}_b)$ have the values

$$\vec{u}_b(\vec{x}_b) = \begin{pmatrix} a_1 sin(2\pi x_b/T) \\ a_2 sin(2\pi y_b/T) \\ a_3 sin(2\pi z_b/T) \end{pmatrix}$$

with

$$T = 30, \;\; a_1 = 9, \;\; a_1 = 8, \;\; a_1 = 12.$$

The results are resumed in table C.2.

As can be seen, the transformation estimation cannot be improved by augmenting the number of iterations. The sinusoidal transformation proves to be difficult to estimate by the B-Spline transformation.

Bibliography

[1] Ourselin S.: *Recalage d'images médicales par appariement de régions.* Thèse de Doctorat, Inria Sophia Antipolis, 2002.

[2] Ourselin S.; RocheA.; Prima S.; Ayache N.: *Block Matching: A General Approach to Improve Robustness of Rigid Registration of Medical Images.* Third International Conference on Medical Image Computing And Computer-Assisted Intervention (MICCAI), pages 557-566, Pittsburgh, Pennsylvania, 2000.

[3] Handels H.: *Medizinische Bildverarbeitung.* Leitfaeden der Informatik, B.G. Teubner Verlag, Stuttgart, 2000.

[4] Maintz A.; Viergever M.: *An Overview of Medical Image Registration Methods* In Symposium of the Belgium hospital physicists association, volume 12, pages V:1-22, 1996/1997.

[5] Hainal J.; Hill D.; Hawkes D.: *Medical Image Registration.* The Biomedical Engineering Series, CRC Press LLC, Florida, 2001.

[6] Pennec X.: *L'Incertitude dans les Problèmes de Reconnaissance et de Recalage.* Thèse de Doctorat, Inria Sophia Antipolis, 1996.

[7] Brown L.G.: *A Survey of Image Registration Techniques.* ACM Computing Surveys, 24(4): 325-376, 1992.

[8] Cachier P.: *Recalage Non Rigide d'Images Medicales Volumiques.* Thèse de Doctorat, Inria Sophia Antipolis, 2002.

[9] Pitiot A.; Bardinet E.; Thompson P.M.; Malandain G.:*Automated Piecewise Affine Registration of Biological Images.* Lecture Notes in Compute Sciences, Biomedical Image Registration, Online Date 2003.

[10] Chen Y.S.; Hung Y.P.; Fuh C.S.: *Fast Block Matching Algorithm Based on the Winner-Update Strategy.* IEEE Transactions on Image Processing, Vol.10, No.8, 2001.

[11] Ourselin S.; Roche A.; Subsol G. Pennec X.; Sattonnet C.: *Automatic Alignment of Histological Sections for 3D Reconstructions and Analysis.* Analytical Cellular Pathology Journal, 19(3):123 1999.

[12] Malandain G.: *Baladin: mode d'emploi.* Manual, Epidaure Project, INRIA Sophia Antipolis, 2003.

[13] Jaehne B.: *Digitale Bildverarbeitung.* Springer Verlag Berlin, Heidelberg, New York, 2002.

[14] Hoschek J.; Lasser D.: *Fundamentals of Computer Aided Geometric Design.* B.G. Teubner, Stuttgart, 1989.

[15] Unser M.: *Splines: A Perfect Fit for Signal and Image Processing.* IEEE Signal Processing Magazine, November 1999.

[16] Kybic J.: *Elastic Image Registration using Parametric Deformation Models.* Thèse de Doctorat, Ecole Polytechnique de Lausanne, 2001.

[17] Rueckert D.: *Non-Rigid Registration: Concepts, Algorithms and Applications.* in The Biomedical Engineering Series, CRC Press LLC, Florida, 2001.

[18] Rueckert D.; Sonoda, L.I.; Hayes, C.; Hill, D.L.G.; Leach, M.O.; Hawkes, D.J.: *Nonrigid Registration Using Free-Form Deformations: Applications to Breast MR Images.* IEEE Transaction on Medical Imaging, Vol. 18, No 8, August 1999.

[19] Dietz U.: *B-Spline Approximation with Energy Constraints.* In Reverse Engineering, B.G. Teubner, Stuttgart, 1996.

[20] Declerck J.: *Etude de la Dynamique cardiaque par Analyse d'Images tridimension-nelles.* Thèse de Doctorat, Inria Sophia Antipolis, 1997.

[21] Van den Elsen P.; Pol E.; Sumanaweera T.; Hemler P.; Napel S.; Adler J.: *Grey value correlation techniques used for automatic matching of Ct and MRT volume images of the head. SPIE vol.2359, pp 227-237,1993.*

[22] Thévenaz P.; Ruttimann U.; Unser M.: *A Pyramid Approach to Subpixel Registration Based on Intensity*. IEEE Transaction on Image Processing 7(1):27-41, 1998.

[23] Lee S.; Wolberg G.; Shin S.Y.: *Scattered Data Interpolation with Multilevel B-Splines*. IEEE Transactions on Visualization and Computer Graphics, Vol.3, No.3, 1997.

[24] Kybic J.; Unser M.: *Fast Parametric Elastic Image Registration*. IEEE Transactions on Image Processing, Vol.12, No.11, 2003.

[25] Meister A.: *Numerik linearer Gleichungssysteme - Eine Einfuehrung in moderne Verfahren*. Vieweg Verlag, Wiesbaden, 1999.

[26] Shewchuk J.R.: *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Article, School of Computer Science, Carnegie Mellon University, Pittsburgh, August 1994.

[27] Bjoerck A.: *Numerical Methods for Least Squares Problems*. SIAM, 1996.

[28] Saad Y.:*Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, 1996.

[29] Risler J.-J.: *Méthodes mathématiques pour la CAO*. Masson, 1991.