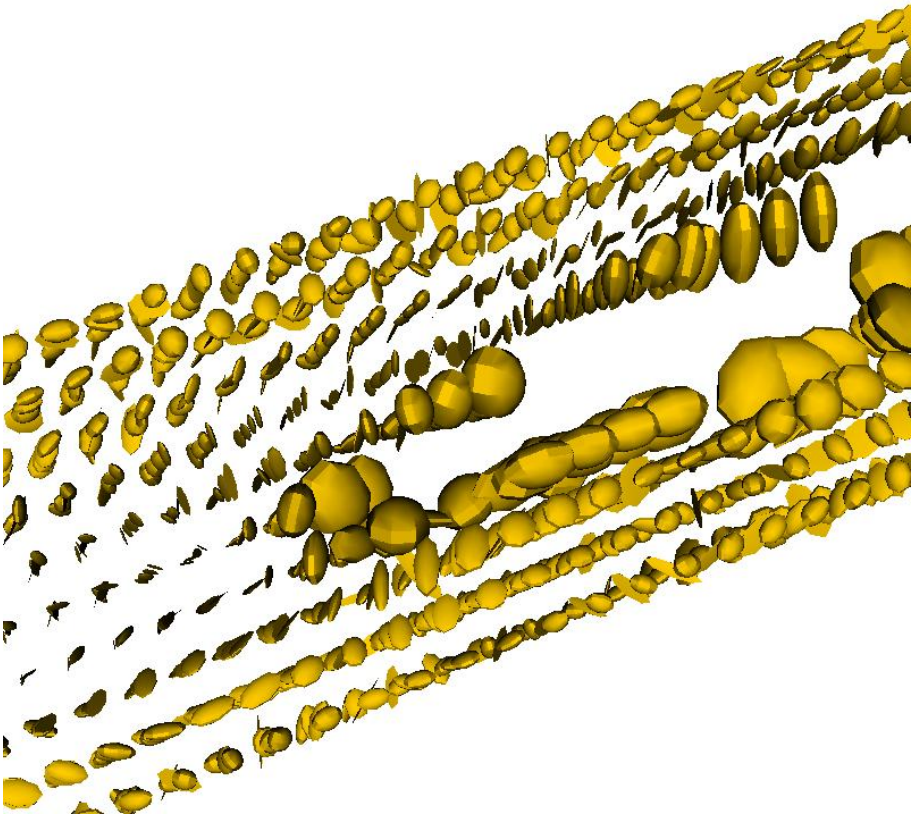

Nonrigid registration of diffusion tensor images

Raimundo Sierra
Electrical Engineering, Master Thesis
rsierra@osola.com

Advisor at ETH Zürich: Prof. Dr. Gabor Székely
Advisor at Harvard: Simon K. Warfield, Ph.D.

March 2001

Swiss Federal Institute of Technology (ETHZ)
Computer Vision Laboratory
Medical Image Analysis and Visualization Group
Gloriastrasse 35
ETH-Zentrum
CH - 8092 Zürich, Switzerland
http://www.vision.ee.ethz.ch/

Brigham and Women's Hospital
Surgical Planning Laboratory
75 Francis Street
Boston, Massachussetts, USA
http://splweb.bwh.harvard.edu:8000/

Frontpage image:
Diffusion tensors around the anterior horn of the
lateral ventricel in an adult human brain

Abstract

This thesis discusses diffusion tensor imaging as a Magnetic Resonance Imaging modality. Diffusion tensor imaging allows the observation of molecular diffusion in tissues *in vivo* and therefore the molecular organization in tissues. The main interest in this case is the observation of myelinated fibertracts in the brain of premature born babies. Myelination of fibers in the white matter of the brain is a fast process in the last few weeks preterm and the observation of this process gives an insight in the development of the human brain and allows a better and earlier detection of small injuries or abnormalities.

The goal is to match diffusion tensor images of neonates, and to build an enabling technology to ultimately generate a statistical atlas of the development of the brain in babies between 28 and 40 weeks postconceptional age. While it was not possible to build the statistical atlas in the time given, the complete process from preprocessing of the data to nonrigid alignment of diffusion tensor images has been implemented and successfully applied on some exemplary cases.

To better understand the characteristics of diffusion tensors and to be able to prove the correctness of the algorithms, a new way of displaying diffusion tensors was implemented. This method visualizes the diffusion tensor as ellipsoids in a voxel raster.

The following report outlines the medical background, the imaging acquisition process and the data processing path. The reader should be able to understand diffusion tensor imaging and the matching principles used and expand the provided software to fit specific and further needs.

To successfully build a meaningful atlas of the development of the brain in neonates, a number of three-dimensional cases needs to be processed and a statistical analysis of the results has to be performed. Therefore a correct incorporation of the different voxel dimensions has to be implemented. As the data quality of the diffusion tensor images in baby scans is very low, the incorporation and combination with other scanning modalities should be considered.

Acknowledgments

First, I would like to thank Prof. Gabor Székely at the Swiss Federal Institute of Technology in Zürich, Switzerland, for his support and motivation for a diploma thesis in the USA, at the Surgical Planning Laboratory of Brigham and Women's Hospital, Boston, Massachusetts. A very special thanks goes to my supervisor at the Surgical Planning Laboratory, Dr. Simon Warfield, who always had time to answer any questions and discuss all the problems that arose. I also want to thank Ron Kikinis, head of the Surgical Planning Laboratory, who made this stay possible.

Furthermore, I want to say thanks to everybody at the Surgical Planning Laboratory who helped me get started with the different subjects I had to learn about, especially Carl-Fredrik Westin, Steven Haker, Hatsohu Mamata, Gary Zientara and Stephan Maier.

Of course, there are lots of other people at the SPL who made this work a great experience.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The development of the human brain is probably one of the most interesting topics in the observation of neonates and premature infants. Definition of anatomical and temporal characteristics of development of critical brain structures in the living premature infant is crucial for an insight into the time of greatest vulnerability in such structures.

Diffusion tensor magnetic resonance imaging is the only noninvasive means available today that approaches the molecular diffusion process *in vivo* and therefore allows the observation of the microstructural development in the human newborn cerebral white matter.

Different studies have been performed to analyze this development quantitatively [2] [6]. Other studies use this imaging modality for an early detection of small injuries [4] [5].

The inter-subject comparison of this microstructural development will make it possible to build a statistical atlas for the normally developing brain. Comparing this atlas to single subjects could allow the early detection of injuries or abnormalities. A first step to build such an atlas is the rigid and nonrigid registration of two different subjects to remove natural inter-subject differences. This work presents a method on how to match tensor images, which represent the molecular diffusion process, of different subjects.

- Chapter 2 gives an insight on the medical application for which this work was done.
- Chapter 3 explains how diffusion tensor images are acquired with magnetic resonance technologies and the characteristics of human brain in this image modality.
- Chapter 4 discusses some general characteristics of the tensor data structure, different measurements that can be applied, and finally some ways of visualizing tensor data.
- Chapter 5 to 7 present the data processing path from the scanner to the nonrigid matching of two different data sets. This path is illustrated in Figure 1.1. Chapter 5 describes the preprocessing which entails the conversion of the scanner output into tensor data, i.e. solving the equations presented in Chapter 3 as well as the generation of the corresponding tensor data structure. In Chapter 6 the process of rigidly aligning two images (and specifically tensor data) is described. Chapter 7 finally presents the nonrigid registration of two data sets and all the methods involved.
- Chapter 8 is a short presentation of some ways of segmenting diffusion tensor data. It should be mentioned that this is only a description of the segmentations that can easily be performed with the current implementation, but there is no further investigation on the quality and meaning of the segmentation.
- Chapter 9 gives an overview of the program structure, the interaction and function of the different classes and the data representation. Appendix A includes a description of the most important functions.

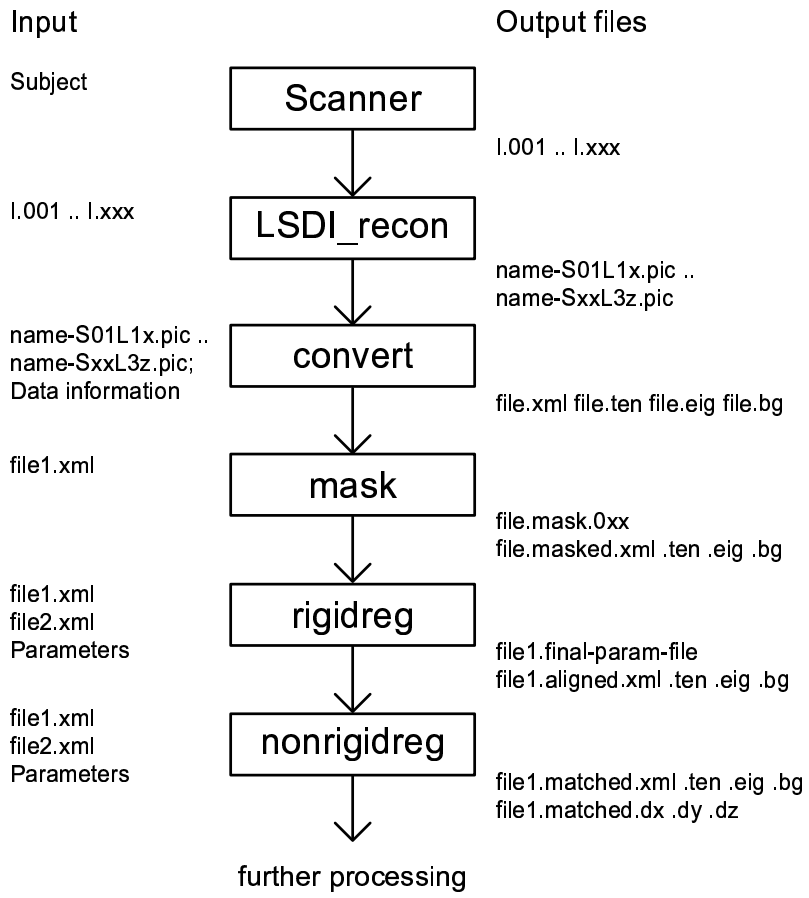

Figure 1.1: Data processing path

- Chapter 10 presents some conclusions and discusses different extensions and improvements that should be considered.

Through this report D will denote a tensor which is a symmetric 3×3 matrix. Positions in an image or volume Ω of any type are indexed with i, j, k for the x, y and z dimensions respectively, so that $D(i, j, k)$ refers to the tensor at position (i, j, k). The dimensions of the volume are X, Y and Z, and the voxel dimensions are δ_X, δ_Y and δ_Z. When comparing two data sets the index S identifies the reference or *s*tationary image, which will not be displaced. The index M refers the *m*oving data set which will be displaced according to a displacement field $U = (v, v, w)^T$ to better match the stationary data set.

A tensorfield denotes a field or volume of tensors which represents the data set. When decomposing the tensor into an eigenvalue, -vector, i.e. an eigensystem, the term eigenfield will be used to describe the data set.

Chapter 2

Medical Background

The registration of different subjects to compare the anatomical variability and statistically analyze this variability is an established process in different medical applications [30].

The basic approach to generating an atlas is to obtain images from a large number of subjects in a mathematical framework that produces a database that is probabilistic. Such an atlas allows the user to obtain relative information that takes into account the variance in structure and function in human populations. Once established such an atlas can interact with new data sets derived from individual subjects and patients [31].

When comparing a new subject with the atlas, abnormalities in this subject are easier to detect. Also the structures in the new subject can be labeled according to the atlas, to which it was registered. Building a temporal atlas allows observation of the mean development in a population.

A quantitative analysis of the brain development in newborns has been done [6]. In this study the total brain volume and total volumes of the cerebral gray matter, unmyelinated white matter, myelinated white matter, and cerebrospinal fluid in premature and mature newborns of postconceptional ages of 29 to 41 weeks have been quantified. The results show that the total brain tissue volume increases linearly at a rate of 22cm^3 per week. Total gray matter shows a linear increase in relative intracranial volume of approximately 1.4% or 15cm^3 in absolute volume per week. The increase in total gray matter is mainly due to a four-fold increase in cortical gray matter.

Unmyelinated white matter is the most prominent brain tissue class in the preterm infant below 36 weeks. Although minimal myelinated white matter is present in the preterm infant at 29 weeks, between 35 and 41 weeks an abrupt five-fold increase in absolute volume of myelinated white matter is documented. The extracerebral and intraventricular cerebrospinal fluid volume changes minimally during this observation period.

Subsequent neurological disability in infants born prematurely and in term infants who experience perinatal hypoxic-ischemic injury is common and serious [7]. In part the tendency for such injuries may relate to a particular vulnerability of actively developing cerebral gray matter and white matter in the last trimester of human gestation. The occurrence of these processes at this maturational time period may render the brain subjected to ischemia or related insults more vulnerable not only to injury but also to subsequent impairment of gray matter and white matter development. The description of the anatomical and temporal characteristics of these developmental processes in the living infant is of great importance in understanding such maturation-dependent vulnerabilities. Quantitative volumetric measurements of cerebral gray and white matter development, including particular myelination, in the living premature and term infant are necessary to describe these anatomical and temporal characteristics.

Another study [2] shows how the water diffusion in certain regions changes over the same period of time (30 to 40 weeks postconceptional age). The results show that the mean apparent diffusion coefficient at 28 weeks is high ($1.8\mu\text{m}^2/\text{ms}$) and decreases towards term ($1.2\mu\text{m}^2/\text{ms}$). Relative

anisotropy is higher the closer birth is to term with greater absolute values in the internal capsule than in the central white matter. Preterm infants at term show higher mean diffusion coefficient in the central white matter (1.4 ± 0.24 versus $1.15 \pm 0.09 \mu m^2/ms$) and lower relative anisotropy in both areas compared to fullterm infants (white matter: 10.9 ± 0.6 versus 22.9 ± 3.0 %; internal capsule $24.0 \pm 4.44\%$ versus $33.1 \pm 0.6\%$). Nonmyelinated fibers in the corpus callosum are visible by diffusion tensor MRI as early as 28 weeks. Fullterm and preterm infants at term show marked differences in white matter fiber organization. The data indicate that quantitative assessment of water diffusion by diffusion tensor Magnetic Resonance Imaging (MRI) provides insight into microstructural development in cerebral white matter in living infants.

The study shows that there are architectural differences between the preterm infants studied at term and the infants born at term. Fiber development and orientation, particularly in the white matter but also in the internal capsule are more evident in the infants born at term. The central white matter in preterm infants at term exhibits less directionality of the diffusion, thinner fiber bundles, and less organized fibers compared to the infants born at term.

A third study [5] shows that it is possible to identify areas of abnormalities with Diffusion Weighted Imaging (DWI) at a time where neither cranial ultrasonography nor conventional MRI detected any definite abnormality. Here the early detection of the diffuse component of the Periventricular Leukomalacia (PVL) with Diffusion Weighted Image is shown.

The findings of the studies presented demonstrate the fields of interest when observing the first possible observation periods of brain development.

Given a diffusion tensor image of a premature born baby (and therefore the stage of myelination of the brain white matter), possible questions when comparing a new subject to an atlas can be:

- What postconceptional age does this subject have?
- Is the development in every structure comparable to the mean development?
- Are there any lesions in the developed structures?
- Are all the expected structures present?

Chapter 3

Image Acquisition

This chapter assumes basic knowledge of magnetic resonance imaging (MRI), i.e. how nuclear magnetic resonance is used to obtain information from the tissue under inspection. Excellent introductions into the principles of MRI are available [8], [9].

The basic idea of diffusion tensor imaging is the same as for Phase Contrast Angiographic MRI. The following sections are mainly a summary of different articles [22].

3.1 Molecular Diffusion and Nuclear Magnetic Resonance

Diffusive transport is observed in steady-state, non-equilibrium systems, such as in cells. A concentration difference is established between two compartments (cells) and a macroscopic diffusive flux can be observed between them. Fick's law describes how the molecular flux density $\mathbf{J}$ depends on the molecular concentration gradient ∇C

$$\mathbf{J} = -D\nabla C \tag{3.1}$$

which leads together with the equation of conservation of mass

$$\frac{\partial C}{\partial t} = -\nabla \mathbf{J} \tag{3.2}$$

to the diffusion equation:

$$\frac{\partial C}{\partial t} = -\nabla \mathbf{J} = \nabla(D\nabla C) \tag{3.3}$$

To be able to determine the diffusivity *in vivo* the diffusion process itself has to be monitored, i.e., the random motions of an ensemble of particles, rather than solving the equation for some initial and boundary conditions. Einstein [10] showed that the diffusion coefficient measured in the non-equilibrium concentration cell experiments is the same quantity that appears in the variance of the conditional probability distribution $P(\mathbf{r}|\mathbf{r}_0, t)$, the probability of finding a molecule at a position $\mathbf{r}$ at a time t which was originally at position $\mathbf{r}_0$. For free diffusion this conditional probability distribution obeys the same diffusion equation as the particle concentration given. The expectancy is then

$$\langle(\mathbf{r} - \mathbf{r}_0)(\mathbf{r} - \mathbf{r}_0)\rangle = 6Dt \tag{3.4}$$

In the case of molecular displacements in tissues, in which diffusion is an anisotropic process with different molecular mobility in x, y and z directions, the diffusion constant D has to be replaced by a diffusion tensor. Equation 3.4 shows that the diffusivity can be inferred directly

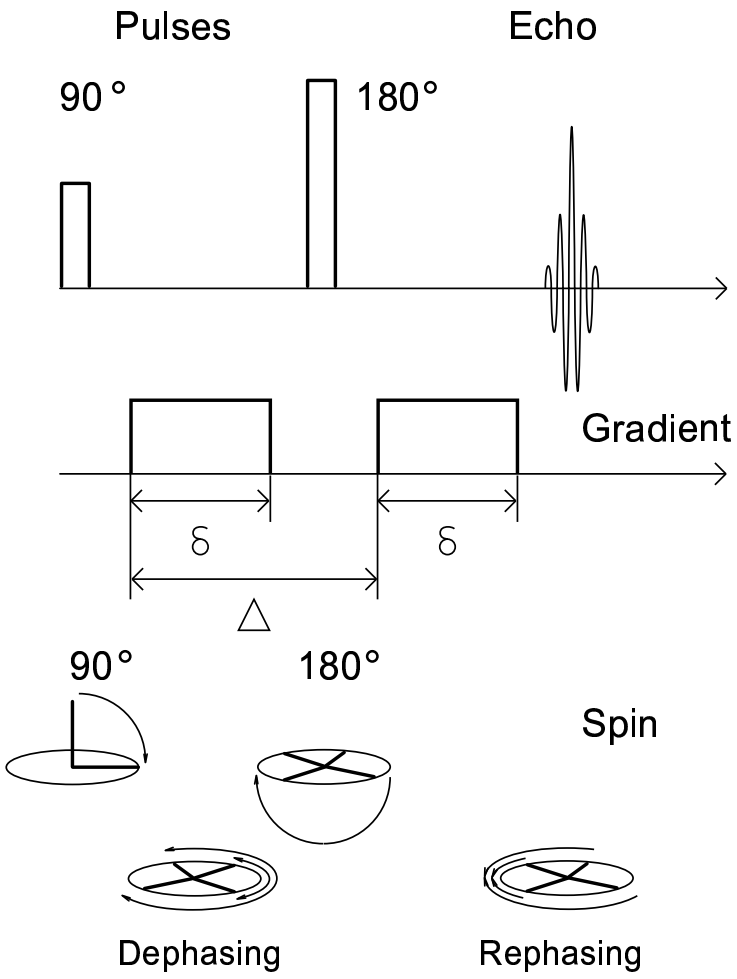

Figure 3.1: Bipolar pulsed gradient experiment (adapted from [22] and [9])

by measuring the second moment of the conditional probability distribution of the diffusing species.

The basic principles of diffusion imaging can be understood from a simple bipolar pulsed gradient experiment (see Figure 3.1). The purpose of these gradient pulses is to magnetically label spins carried by molecules. Here G denotes the gradient strength, δ as the gradient duration and Δ as the time interval between the pulses. The first gradient pulse induces a phase shift ϕ_1 of the spin transverse magnetization, which depends on the spin position. If the gradient is along z, then:

$$\phi_1 = \gamma \int_0^{\delta} G z_1 dt = \gamma G \delta z_1 \tag{3.5}$$

z_1 is the spin position supposed to be constant during the short duration δ of the gradient pulse. γ is the gyromagnetic ratio, which is a nuclear specific factor. Its value is 42 for hydrogen ^{1}H protons. After the 180° radio frequency (RF) pulse, ϕ_1 is transformed into $-\phi_1$. Similarly, the second pulse after a delay δ will produce a phase shift ϕ_2:

$$\phi_2 = \gamma \int_{\Delta}^{\Delta+\delta} G z_2 dt = \gamma G \delta z_2 \tag{3.6}$$

where z_2 is the spin position during the second pulse. The resulting net dephasing $\delta(\phi)$ is:

$$\delta(\phi) = \phi_2 - \phi_1 = \gamma G \delta(z_1 - z_2) \tag{3.7}$$

It can be seen that for static spins, i.e. not moving molecules $z_1 = z_2$, the bipolar gradient pair produces no net dephasing. For moving spins there is a net dephasing that will depend on the spin history during the time interval Δ between the pulses, and which will affect the transverse magnetization. The position of the two gradient pulses in each half of the spin-echo sequence does not matter; it is the time elapsed between them that affects the net phase. In Nuclear Magnetic Resonance (NMR) the total magnetization is measured, the vector sum of the magnetic moments M of the individual nuclei, which may have different motion histories:

$$\frac{M}{M_0} = \sum_{j=1}^{N} e^{i\delta(\phi_j)} \tag{3.8}$$

where M_0 is equilibrium magnetization in the direction of the static applied magnetic field $\mathbf{B}_0$. This sum can be evaluated once the net phase distribution is known. Assuming free diffusion in a homogeneous domain, the probability of finding a spin at position z_1 is a constant. If $P(z_2, z_1, \Delta)dz_2$ is the conditional probability of finding a spin initially at z_1 between positions z_2 and $z_2 + dz_2$ after a time interval Δ, the amplitude attenuation is:

$$\frac{M}{M_0} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{i\gamma G \delta(z_1 - z_2)} P(z_2, z_1, \Delta) dz_1 dz_2 \tag{3.9}$$

For free diffusion in one dimension, the conditional probability is given by:

$$P(z_2, z_1, \Delta) = \frac{1}{\sqrt{4\pi D \Delta}} e^{\frac{-(z_1 - z_2)^2}{4D\Delta}} \tag{3.10}$$

where D is the diffusion coefficient. Combining equations 3.9 and 3.10 leads to:

$$\frac{M}{M_0} = e^{-(\gamma G \delta)^2 D \Delta} \tag{3.11}$$

This last equation relates the measured signal attenuation to the diffusivity, and is the basis for diffusion measurement using NMR.

Taking several pulses and the fact that δ may not be negligible as compared to Δ, into account, Equation 3.10 has to be solved for a general pulse sequence. This leads to the following relation for an isotropic medium in a spin-echo experiment (for a detailed derivation of these equations see [22] p. 8, 9).

$$\frac{M(TE)}{M_0} = e^{-D \int_0^{TE} \mathbf{k}(t)\mathbf{k}(t)dt} \tag{3.12}$$

where

$$\mathbf{k}(t) = \gamma \int_0^t G(t')dt' \tag{3.13}$$

Introducing the gradient factor b

$$b = \int_0^{TE} \mathbf{k}(t)\mathbf{k}(t)dt \tag{3.14}$$

which characterizes the sensitivity of NMR sequences to diffusion, the signal attenuation can be represented by the simpler expression

$$\frac{M(TE)}{M_0} = e^{-bD} \tag{3.15}$$

Tissue	Diffusion coefficient $\cdot 10^{-3}$mm^2/s
CSF	2.94 ± 0.05
Gray matter	0.76 ± 0.03
White matter:	
Corpus callosum	0.22 ± 0.22
Axial fibers	1.07 ± 0.06
Transverse fibers	0.64 ± 0.05

Table 3.1: Diffusion coefficients of water in the human brain [22]

Diffusion is a three-dimensional process. However molecular mobility may not be the same in all directions. This anisotropy may be due to the physical arrangement of the medium or the presence of obstacles that limit diffusion (restricted diffusion) or both. Moreover structures that exhibit anisotropic diffusion at the molecular level can be isotropic at the microscopic level.

As mentioned earlier, in anisotropic diffusion the effective diffusion coefficient is replaced by an effective diffusion tensor. The echo attenuation then becomes

$$\frac{M(TE)}{M_0} = e^{-\sum_{i=1}^{3}\sum_{j=1}^{3} b_{ij} D_{ij}} \tag{3.16}$$

where b_{ij} is a b-matrix and D_{ij} is an effective diffusion tensor. Its diagonal terms D_{xx}, D_{yy} and D_{zz} represent correlations between molecular displacements in the same directions, whereas its off-diagonal terms D_{xy}, D_{xz}, D_{yz} reflect correlations between molecular displacements in orthogonal directions.

To obtain the different diffusion coefficients at each voxel position, different echo and gradient sequences have been proposed [23].

3.2 Diffusion in the Normal Brain - Anisotropic Diffusion in White Matter

The diffusion coefficient of water in tissues was found to be 2-10 times less than that of pure water [24], [25]. This is understandable, given that water molecules are obliged to move tortuously around obstructions presented by fibers, intracellular organelles or macromolecules. In addition, there is a continual exchange between free water molecules and water molecules which spend some of their time associated with the much more slowly moving macromolecules. Diffusion is thus more likely to be "hindered" by random obstacles than strictly "restricted" in close spaces by walls. Table 3.1 shows some diffusion coefficients of water in the human brain. The diffusion in the cerebrospinal fluid (CSF) is similar to that of pure water at the same temperature ($2.5 * 10^{-3}$ mm^2/s @ 37.5° C)

As can be seen from Table 3.1, diffusion in white matter is extremely variable. The value of the diffusion coefficient directly depends on the relative orientation of the fibers and the magnetic field gradients, which is known as "anisotropic diffusion." Water diffusion in gray matter does not exhibit anisotropy or restriction by impermeable walls [26], [27]. White matter on the other hand is extremely anisotropic, the results of the measurements depending on the respective orientation of the myelin fiber tracts and the gradient direction at each different image location. It appears that diffusion coefficients are significantly decreased when the myelin fiber tracts are perpendicular to the direction of the magnetic field gradient used to measure molecular displacements.

Figure 3.2 shows adjacent myelinated fibers and the diffusion of water. The diffusion coefficient measured parallel to the myelin fiber direction $D_{\parallel}$ is about three times larger ($1.2 * 10^{-3}$ mm^2/s) than the diffusion coefficient perpendicular to fibers $D_{\perp}(0.4 * 10^{-3}$mm^2/s).

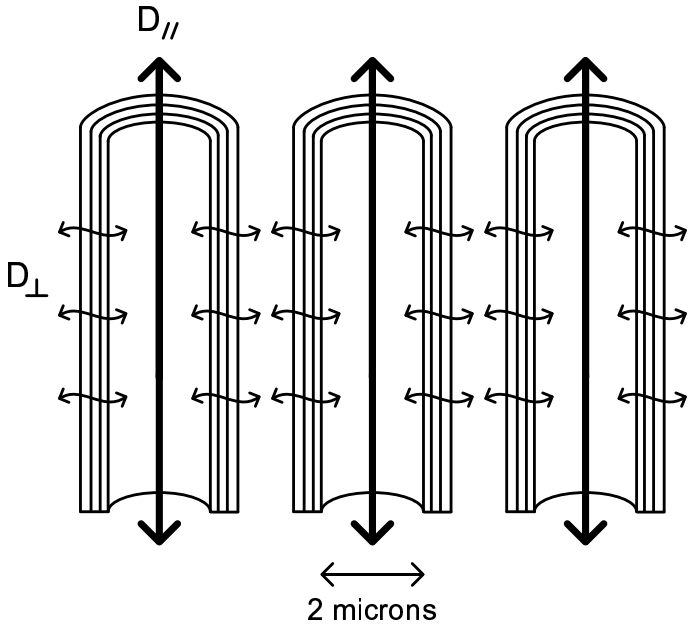

Figure 3.2: Diffusion in myelinated fibers (adapted from [22])

Since MRI methods in general always obtain a macroscopic measure of a microscopic quantity which necessarily entails intravoxel averaging, the voxel dimensions influence the measured diffusion tensor at any particular location in the brain.

Factors which would affect the shape of the apparent diffusion tensor (i.e., the shape of the diffusion ellipsoid) in the white matter include the density of fibers, the degree of myelination, the average fiber diameter, and the directional similarity of the fibers in the voxel. The geometric nature of the measured diffusion tensor within a voxel is thus a meaningful measure of fiber tract organization.

Although the individual axons and the surrounding myelin sheaths cannot be revealed with the limited spatial resolution of *in vivo* imaging, distinct bands of white matter fibers with parallel orientation may be distinguished from others running in different directions. Figure 3.3 shows how two crossing fibertracts would ideally be represented by a diffusion tensor image.

Although there is no doubt that diffusion is anisotropic in white matter, controversies about the origin of this anisotropy remain. The diffusion-time dependence of the measured diffusion coefficient is the crucial experimental test for the presence and dimension of diffusive barriers. If diffusion is restricted by impermeable barriers the diffusion coefficient decreases when the diffusion distance reaches the dimension of the available volume. Water in gray and white matter diffuses without encountering significant barriers - at least on the distance range of 8 - 10 microns, which exceeds the dimensions of most cellular compartments [26]. Anisotropy also exists in brains of neonates before the histological appearance of myelin [18]. This leads to the conclusion that myelination is not essential for the diffusion anisotropy of nerves. Nevertheless myelin is widely assumed to be the major barrier to diffusion in myelinated fiber tracts. Therefore the demonstration of anisotropic diffusion in the brain by magnetic resonance has opened the way to explore noninvasively the structural anatomy of the white matter *in vivo* [16].

In summary, diffusion measurements *in vivo* reflect complicated pathways of water molecules in the tissue.

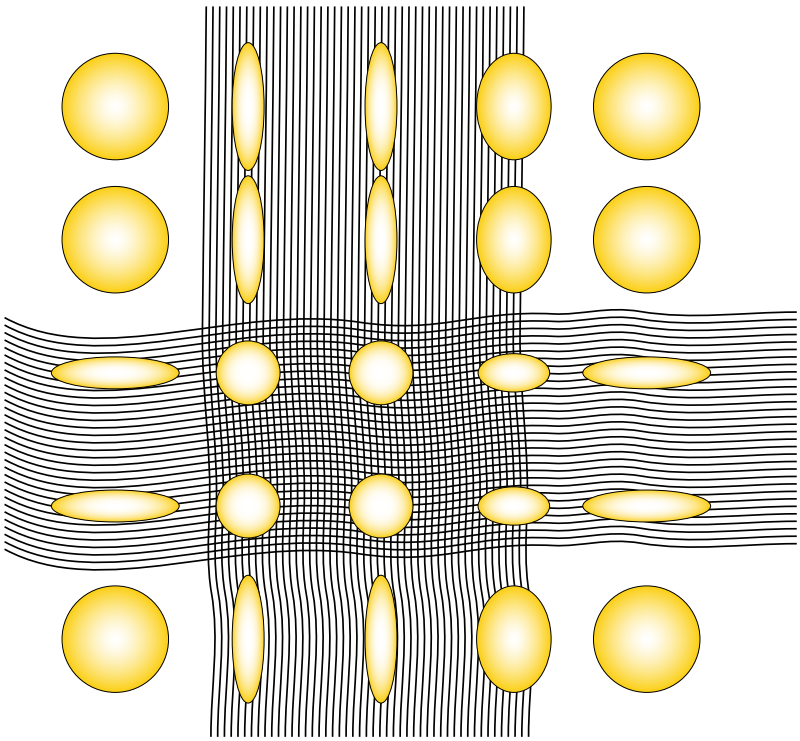

Figure 3.3: Diffusion tensor overlying two crossing fibertracts

3.3 Baby Brain Data

The data used in this study was generated using a modified version of the Line Scan Diffusion Imaging (LSDI) technique [28]. In this technique a bipolar gradient pulse echo is used. This sequence has been shown in Figure 3.1. In this case b in Equation 3.15 becomes

$$b = \gamma^2 G^2 \delta^2 (\Delta - \frac{\delta}{3}) \tag{3.17}$$

so that the loss of signal intensity is (Stejskal Tanner formula)

$$\ln(M) = \ln(M_0) - \gamma^2 G^2 \delta^2 (\Delta - \frac{\delta}{3}) D \tag{3.18}$$

The data was acquired at the Brigham and Women's Hospital on a GE Signa 1.5 Tesla Horizon Echospeed 5.6 system with standard 2.2 Gauss/cm field gradients. The time required for acquisition of the diffusion tensor data for one slice was 1 min; no averaging was performed. Imaging parameters were: effective TR=2.4s, TE=65ms, b_{high}=750 s/mm^2, b_{low} =5 s/mm^2, field of view 18 cm, 6 kHz readout bandwidth, acquisition matrix 128×128.

Usually one coronal and one axial slice with effective voxel dimensions $5 \times 0.703125 \times 0.703125$ mm^3 were acquired. Newer acquisitions (September 1998 to present) with multiple slices (between 9 and 14 slices) have an effective voxel size of $4.4 \times 0.703125 \times 0.703125$ mm^3. Axial and coronal multislice acquisitions locations are chosen to include the majority of the white matter. Figure 3.4 shows roughly in a sagittal view the positions from where axial and coronal, multiple slice diffusion weighted scans were taken. The factor of in-plane to inter-slice resolution is still very large (about 7.1 respectively 6.25) so that most processing is done separately for each slice.

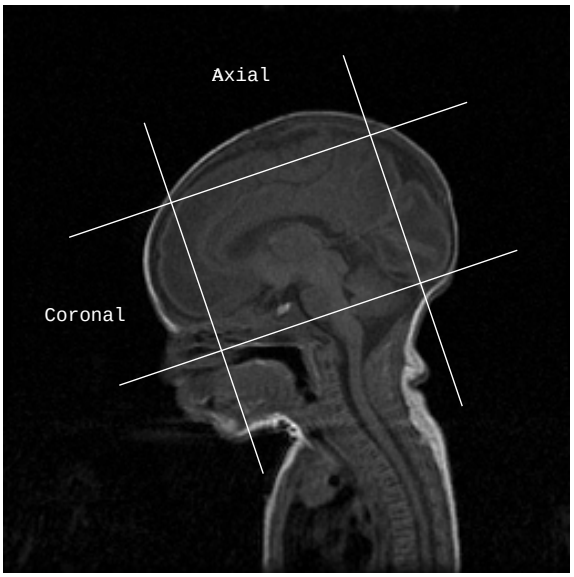

Figure 3.4: Sagittal view showing the positions from where axial and coronal, multiple slice diffusion weighted scans were taken.

Chapter 4

Tensors

In this chapter the data structure of a tensor is studied. First a mathematical insight of tensor data is given. Some measurements related to tensor characteristics are presented, followed by different ways of applying transformations on tensors. Finally the visualization of a tensor is discussed.

4.1 Tensor Characteristics

A scalar is a quantity whose specification (in any coordinate system) requires just one number. A tensor of order n on the other hand is an object that requires 3^n numbers in any given coordinate system. With this definition, scalars and vectors are special cases of a tensor. Scalars are tensors of order 0 with $3^0 = 1$ components, and vectors are tensors of order 1 with $3^1 = 3$ components. The diffusion tensors are general tensors of order 2 with $3^2 = 9$ components. The components of a second order tensor are often written as a 3×3 matrix, as will be done here. Moreover the diffusion tensor is a symmetric second order tensor so that the matrix is of the form

$$D = \begin{pmatrix} D_{11} & D_{12} & D_{13} \\ D_{12} & D_{22} & D_{23} \\ D_{13} & D_{23} & D_{33} \end{pmatrix} \tag{4.1}$$

A tensor can be reduced to principal axes (eigenvalue and eigenvector decomposition) if the equation

$$D\mathbf{e} = \lambda\mathbf{e} \tag{4.2}$$

or

$$(D - \lambda I)\mathbf{e} = 0 \tag{4.3}$$

where I is the identity matrix, has a nontrivial solution.

Let $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$ be the eigenvalues of the symmetric tensor D and let $\hat{\mathbf{e}}_i$ be the normalized eigenvector corresponding to λ_i. As the diffusion tensor is symmetric the eigenvalues λ_i will always be real. Moreover the corresponding eigenvectors are perpendicular.

The values λ_i can be found by solving the characteristic equation

$$\begin{vmatrix} D_{11} - \lambda & D_{12} & D_{13} \\ D_{21} & D_{22} - \lambda & D_{23} \\ D_{31} & D_{32} & D_{33} - \lambda \end{vmatrix} = 0 \tag{4.4}$$

or

$$\lambda^3 - \lambda^2(D_{11} + D_{22} + D_{33}) + \lambda\left(\begin{vmatrix} D_{22} & D_{32} \\ D_{23} & D_{33} \end{vmatrix} + \begin{vmatrix} D_{11} & D_{21} \\ D_{12} & D_{22} \end{vmatrix} + \begin{vmatrix} D_{11} & D_{31} \\ D_{13} & D_{33} \end{vmatrix}\right) - \begin{vmatrix} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{vmatrix} = 0 \quad (4.5)$$

after expanding the determinant. The numbers λ (scalars) are independent of the choice of the coordinate system and hence so are the coefficients in Equation 4.5.

Therefore the quantities

$$\begin{aligned} I_1 &= D_{11} + D_{22} + D_{33} \\ I_2 &= \begin{vmatrix} D_{22} & D_{32} \\ D_{23} & D_{33} \end{vmatrix} + \begin{vmatrix} D_{11} & D_{21} \\ D_{12} & D_{22} \end{vmatrix} + \begin{vmatrix} D_{11} & D_{31} \\ D_{13} & D_{33} \end{vmatrix} \\ I_3 &= \begin{vmatrix} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{vmatrix} \end{aligned} \quad (4.6)$$

are all invariants of the tensor D. Using these invariants one can form infinitely many other invariants. I_1 is known as the trace, I_3 is the determinant of D.

The inverse transformation from an eigensystem to a tensor is given by

$$D = [\hat{\mathbf{e}}_1\ \hat{\mathbf{e}}_2\ \hat{\mathbf{e}}_3]^{-1} \cdot diag\,[\lambda_1\ \lambda_2\ \lambda_3] \cdot [\hat{\mathbf{e}}_1\ \hat{\mathbf{e}}_2\ \hat{\mathbf{e}}_3] \quad (4.7)$$

In the case of a symmetric tensor this equation simplifies to

$$D = \lambda_1\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^T + \lambda_2\hat{\mathbf{e}}_2\hat{\mathbf{e}}_2^T + \lambda_3\hat{\mathbf{e}}_3\hat{\mathbf{e}}_3^T \quad (4.8)$$

where the eigenvectors $\hat{\mathbf{e}}_i$ form an orthonormal basis.

4.2 Measurements

Using this decomposition, diffusion can be divided into three basic cases depending on the rank of the tensor [16]:

1. Linear case ($\lambda_1 \gg \lambda_2 \simeq \lambda_3$): Diffusion is mainly in the direction of the eigenvector of the largest eigenvalue:
$$D \simeq \lambda_1 D_l = \lambda_1\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^T \quad (4.9)$$

2. Planar case ($\lambda_1 \simeq \lambda_2 \gg \lambda_3$): Diffusion is mainly in the plane spanned by the two eigenvectors corresponding to the two largest eigenvalues:
$$D \simeq 2\lambda_1 D_p = \lambda_1\left(\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^T + \hat{\mathbf{e}}_2\hat{\mathbf{e}}_2^T\right) \quad (4.10)$$

3. Spherical case ($\lambda_1 \simeq \lambda_2 \simeq \lambda_3$): Diffusion is isotropic in all directions:
$$D \simeq 3\lambda_1 D_s = \lambda_1\left(\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^T + \hat{\mathbf{e}}_2\hat{\mathbf{e}}_2^T + \hat{\mathbf{e}}_3\hat{\mathbf{e}}_3^T\right) \quad (4.11)$$

A general diffusion tensor D will be a combination of these cases. Expanding the diffusion tensor using these base cases gives:

$$\begin{aligned} D &= \lambda_1\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^T + \lambda_2\hat{\mathbf{e}}_2\hat{\mathbf{e}}_2^T + \lambda_3\hat{\mathbf{e}}_3\hat{\mathbf{e}}_3^T \\ &= (\lambda_1 - \lambda_2)\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^T + (\lambda_2 - \lambda_3)\left(\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^T + \hat{\mathbf{e}}_2\hat{\mathbf{e}}_2^T\right) + \lambda_3\left(\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^T + \hat{\mathbf{e}}_2\hat{\mathbf{e}}_2^T + \hat{\mathbf{e}}_3\hat{\mathbf{e}}_3^T\right) \\ &= (\lambda_1 - \lambda_2)D_l + (\lambda_2 - \lambda_3)D_p + \lambda_3 D_s \end{aligned} \quad (4.12)$$

where $(\lambda_1 - \lambda_2)$, $(\lambda_2 - \lambda_3)$ and λ_3 are the coordinates of D in the tensor basis D_l, D_p, D_s. This relation between the eigenvalues of the diffusion tensor can be used to classify the diffusion tensor according to a geometrically meaningful criteria. By using the new basis for the tensor, measures are obtained of how close the diffusion tensor is to the generic cases of line, plane and sphere. The generic shape of a tensor is obtained by normalizing with a magnitude measure of the diffusion. A useful measure in this context is the magnitude of the largest eigenvalue of the tensor, normalizing the sum of the measurements to 1:

$$c_l = \frac{\lambda_1 - \lambda_2}{\lambda_1} \tag{4.13}$$

$$c_p = \frac{\lambda_2 - \lambda_3}{\lambda_1} \tag{4.14}$$

$$c_s = \frac{\lambda_3}{\lambda_1} \tag{4.15}$$

$$c_l + c_p + c_s = 1 \tag{4.16}$$

An anisotropy measure describing the deviation from the spherical case is:

$$c_a = c_l + c_p = 1 - c_s = 1 - \frac{\lambda_3}{\lambda_1} \tag{4.17}$$

Smoothing In image processing a common operation is smoothing of the data to reduce the noise level. For diffusion data, an independent smoothing of the tensor components has proven to be a robust method. Figure 4.2 shows the effect of applying a Gauss filter to each component of the field in Figure 4.1. Figure 4.3 is a field where the tensors have a clear bias in one direction (the maximum angle between the eigenvectors corresponding to the largest eigenvalue was set to 10°). When smoothing this field, the bias is clearly preserved (see Figure 4.4). This form of smoothing also allows to perform the computation in the tensor-domain.

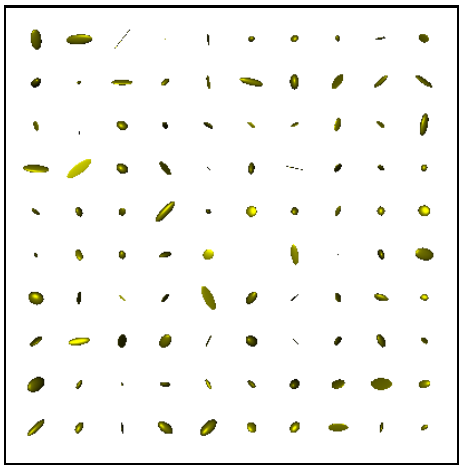

Figure 4.1: Random tensorfield

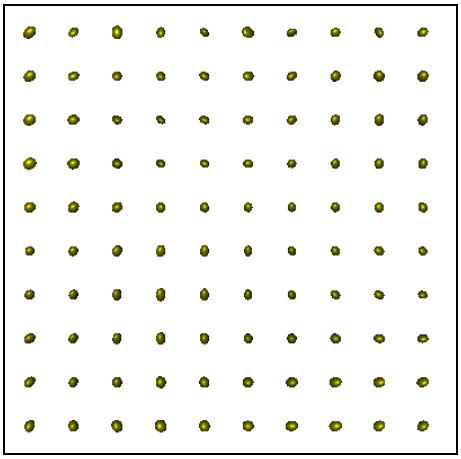

Figure 4.2: Smoothing with Gauss filter of length 5

Interpolation Similarly the transition from one tensor to another, i.e. the interpolation of tensors, is also performed on the single components of the tensor which leads to a result as shown in Figure 4.5. This interpolation is certainly the safest if no information of the structure between the tensors is available. But it is not the only possible transition. Figure 4.6 shows a transition

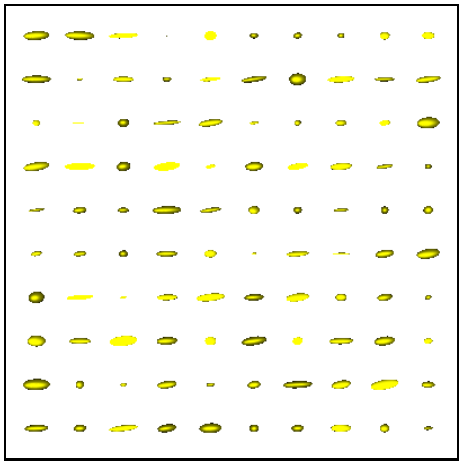

Figure 4.3: Random tensorfield with bias in one direction

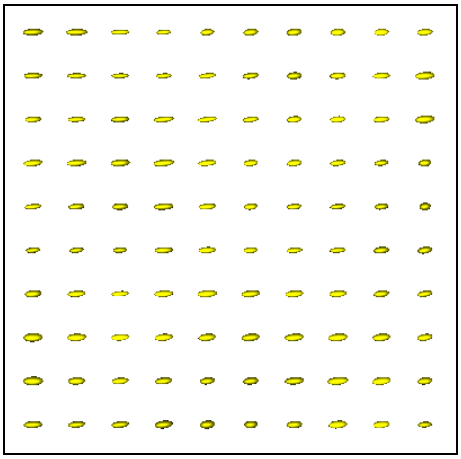

Figure 4.4: Smoothing with Gauss filter of length 3

between two tensors where the shape of the tensor is preserved during the transformation. This form will be further discussed in Section 7.4 on page 34, when the displacement of a tensorfield is discussed.

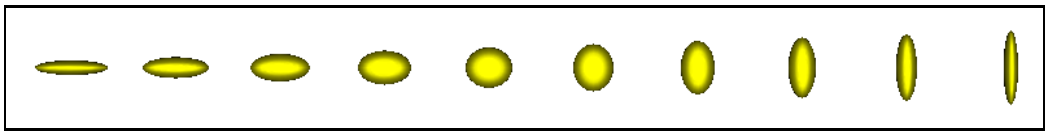

Figure 4.5: Interpolation of single components of two tensors in 9 steps

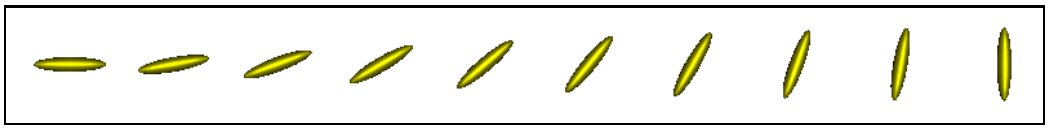

Figure 4.6: Interpolation of two tensors in 9 steps by keeping the shape

4.3 Visualization

Unlike scalar data the tensor is a three dimensional structure at each voxel position. Therefore simple grayscale images are not suitable for the representation of tensor data. Some ways of displaying tensor fields are presented and discussed here.

The tensor can be represented as an ellipsoid where the main axes lengths correspond to the eigenvalues and their direction to the respective eigenvectors. This method of display has already been used to illustrate the tensor characteristics in the previous section.

However, when displaying tensors as ellipsoids, there is no difference between an edge-on, flat ellipsoid, and an oblong one, or between a face-on, flat ellipsoid, and a sphere. By assigning a specular intensity and power to the ellipsoids, the reflection of the light source gives an insight of the third dimension of the ellipsoid. This allows a distinction between the ambiguous cases

mentioned. Also the field can be rotated in all three dimensions so that the ellipsoids can be inspectioned from any direction.

In the implementation, the tensors are classified into three different classes depending on their shape, and color-encoded according to the class they belong to. The tensors are assigned to a class depending on how close to a line, plane or sphere they are:

$$class = \begin{cases} 0 & : \quad c_l \geq c_p, c_s \\ 1 & : \quad c_p \geq c_l, c_s \\ 2 & : \quad c_s \geq c_l, c_p \end{cases} \tag{4.18}$$

Figure 4.7 illustrates this displaying technique, where class 0 tensors are displayed in blue, class 1 tensors in yellow and class 2 tensors in yellow with the transparency set to 0.4. More example

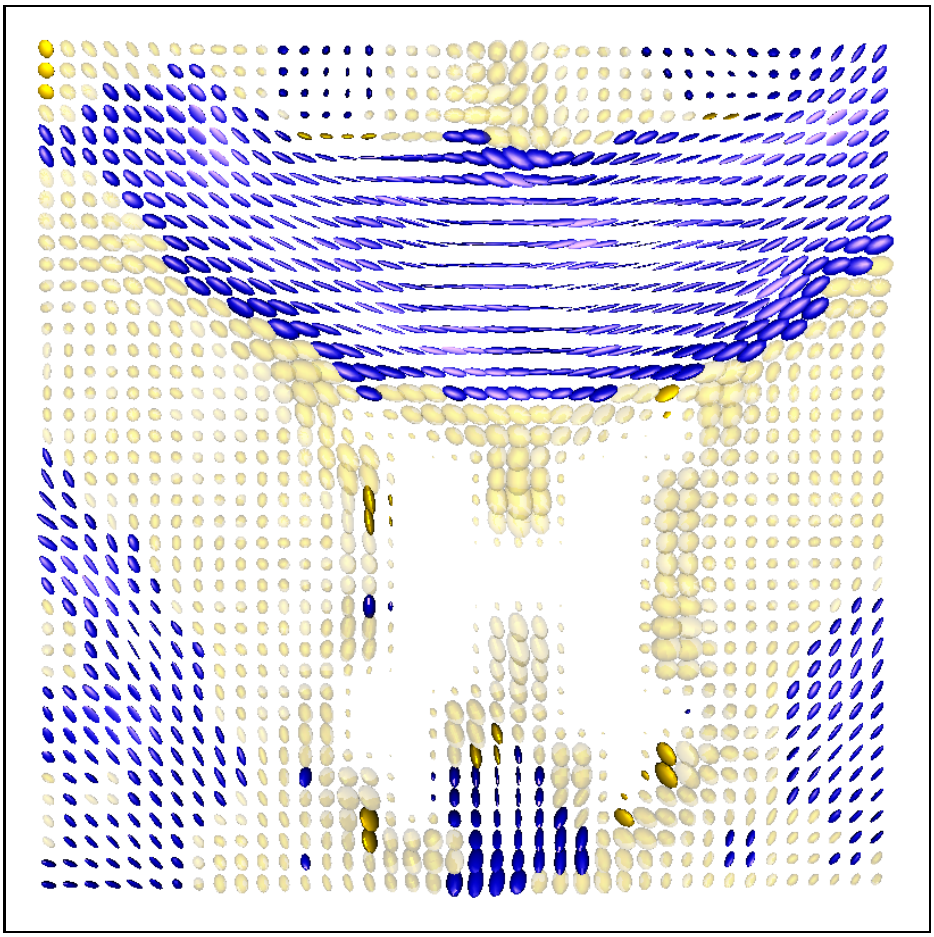

Figure 4.7: Visualization of tensors using ellipsoids. Example showing the corpus callosum of an adult human brain

images using this representation of tensors can be found in Appendix B.

The representation of a tensorfield in ellipsoids is certainly limited in size, since the overall information provided is too large for a visual inspection. Therefore two dimensional representations of the tensors are useful when observing larger objects.

One way of visualizing the tensors as two-dimensional objects it to use blue headless arrows that represent the in-plane components of $c_l\hat{\mathbf{e}}_1$ [17]. The out-of-plane components of $c_l\hat{\mathbf{e}}_1$ are shown

in colors ranging from green through yellow to red, with red indicating the highest value for this component. Figure 4.8 shows the example slice using this visualization technique.

Figure 4.9 shows another way of color encoding the eigenvector corresponding to the largest eigenvalue. Here the components of the first eigenvector $\hat{\mathbf{e}}_{1x}$, $\hat{\mathbf{e}}_{1y}$ and $\hat{\mathbf{e}}_{1z}$ are multiplied by the length of the eigenvalue λ_1. The red, green and blue (RGB) values for the color at a position (i, j, k) are then set to

$$\begin{aligned} R &= k\lambda_1\hat{\mathbf{e}}_{1x} \\ G &= l\lambda_1\hat{\mathbf{e}}_{1y} \\ B &= m\lambda_1\hat{\mathbf{e}}_{1z} \end{aligned} \tag{4.19}$$

with k, m, l parameters to scale each component into a range from 0 to 255.

The disadvantage of this form of visualization is that fibertracts change color when changing direction, even though these color changes will be smooth. Nevertheless, it is a useful visualization to get an impression of the quality and the content of the data set.

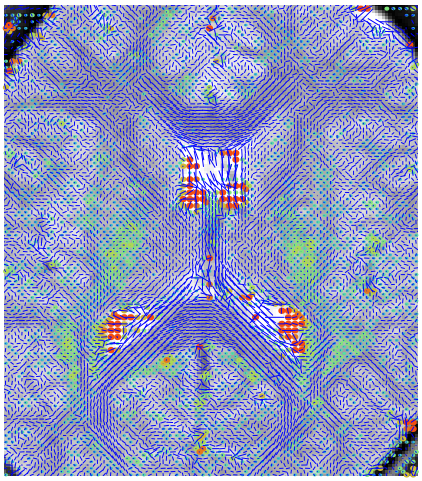

Figure 4.8: Visualizing tensors with headless arrows and dots representing $c_l\hat{\mathbf{e}}_1$

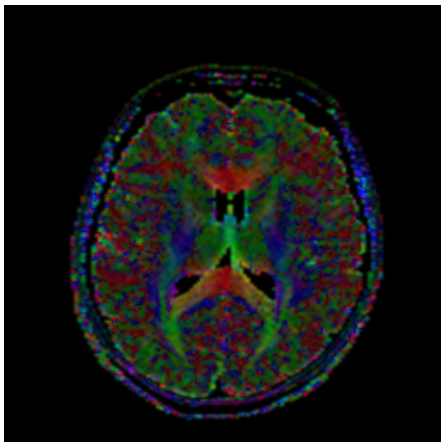

Figure 4.9: Visualizing tensors by color-encoding the largest eigenvalue, -vector

Finally the different measurements presented in Section 4.2 can be represented as grayscale images. Figure 4.10, 4.11 and 4.12 show this measurements for the same subject.

The program `main` allows the display of any tensorfield which has been preprocessed as described in Chapter 5. The displaying of tensor data sets with ellipsoids is a very computation intensive process. It is therefore recommended that in displaying an image that one begins with a very small data window to see if the result is presented in a reasonable time.

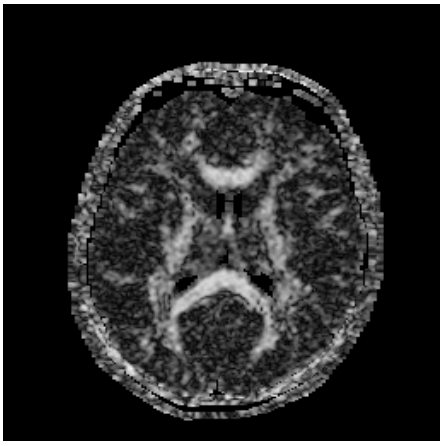

Figure 4.10: Grayscale representation of the measurement close-line c_l

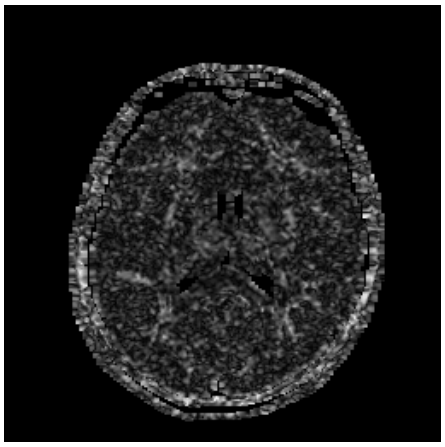

Figure 4.11: Grayscale representation of the measurement close-plane c_p

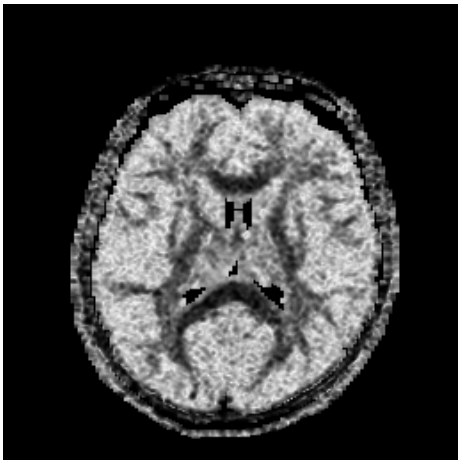

Figure 4.12: Grayscale representation of the measurement close-sphere c_s

Chapter 5

Data Preprocessing

This chapter describes how the data is processed from output of the scanner to the actual diffusion tensor representation.

5.1 Solving the Basic Equations

`LSDI_recon` is a program that solves the Stejskal Tanner Equation 3.18 presented in Section 3.3. As input the program requires the images produced by the scanner as well as a parameter file. The eight images provide eight equations for M in each voxel which are solved in a least-squares sense for the 6+1 unknowns: the six independent components of the symmetric diffusion tensor D and M_0. Besides solving the equations, `LSDI_recon` reorients and interpolates the diffusion tensor data so that the resulting output has a standardized orientation and shape. The output is saved as `*.pic` files of size 256×256 with short precision and represents the eigenvalue and eigenvector decomposition of the tensor. There is no header and therefore no additional information of the data is stored. Table 5.1 describes the content of the single images that are produced by `LSDI_recon`.

File extension	Content
file-SxxL1e.pic	first eigenvalue
file-SxxL1x.pic	first eigenvector
file-SxxL1y.pic	
file-SxxL1z.pic	
file-SxxL2e.pic	second eigenvalue
file-SxxL2x.pic	second eigenvector
file-SxxL2y.pic	
file-SxxL2z.pic	
file-SxxL3e.pic	third eigenvalue
file-SxxL3x.pic	third eigenvector
file-SxxL3y.pic	
file-SxxL3z.pic	
file-SxxAAI.pic	Absolute anisotropy index
file-SxxADC.pic	Apparent diffusion coefficient
file-SxxDWI.pic	Diffusion weighted image
file-SxxRAI.pic	Relative anisotropy = $\frac{\text{AAI}}{\text{ADC}}$
file-SxxT2W.pic	T2 weighted image

Table 5.1: File extensions and content of the files generated by `LSDI_recon`. xx represents the slice number.

5.2 Conversion to Tensor Fields

In this step the files generated by `LSDI_recon` are converted into floating point precision data structures representing tensor- and eigenfields. Additional images produced by `LSDI_recon` are considered as background images and are also collected into a single data structure. An additional xml-file containing information about the data is generated. The xml-file and its Document Type Definition (DTD) are described in Section 9.3. Most important it stores the field dimension, voxel dimensions and the location of the data without any file extension. Table 5.2 describes the file extensions used and their meaning. Some of the files are generated in later steps and described when they first appear.

File extension	Content
.xml	the xml-header with information of the data
.ten	the tensorfield data
.eig	the eigenfield data
.bg	the background images
.mask.0xx	the mask for the data (MRI file format)
.masked.xml, .ten, .eig, .bg	masked data
.final-param-file	text file with parameters for the rigid transformation
.aligned.xml, .ten, .eig, .bg	the aligned data
.matched.xml, .ten, .eig, .bg	the matched data
.matched.dx	displacement field in x-direction
.matched.dy	displacement field in y-direction
.matched.dz	displacement field in z-direction

Table 5.2: File extensions and content of the files generated and used in this application

5.2.1 Orthogonalization

Despite the fact that `LSDI_recon` does a lot of enhancement of the data the output cannot directly be used as tensor data. Most importantly the simpler Equation 4.8 for conversion from an eigensystem decomposition to a matrix cannot be applied since it was found that the eigenvectors produced by `LSDI_recon` do not build an orthonormal basis. As the tensor has to be symmetric it would be incorrect to apply the general Equation 4.7 to build a matrix from the eigensystem. Making the matrix orthogonal after transformation by computing $D' = 0.5(D + D^T)$, which is the best approximation without any prior knowledge when using the norm $\|D\|^2 = \sum_{i,j}(D_{ij})^2$, deforms the tensor in an unwanted manner as the off-diagonal elements often have opposite signs.

To built an orthonormal basis with the eigenvectors given, it is assumed that the signal-to-noise ratio is best for the largest eigenvalue and therefore also for the corresponding eigenvector. It then makes sense to keep the eigenvector corresponding to the largest eigenvalue and correct the remaining two eigenvectors. The second eigenvector is projected into the plane normal to the first eigenvector. This plane is described by

$$\hat{\mathbf{e}}_{1x}x + \hat{\mathbf{e}}_{1y}y + \hat{\mathbf{e}}_{1z}z = 0 \tag{5.1}$$

where $\hat{\mathbf{e}}_1$ is the first eigenvector. The projection into the plane is then given by

$$\hat{\mathbf{e}}'_2 = s(\alpha\hat{\mathbf{e}}_1 + \hat{\mathbf{e}}_2) \tag{5.2}$$

where

$$\alpha = -\frac{\hat{\mathbf{e}}_{1x}\hat{\mathbf{e}}_{2x} + \hat{\mathbf{e}}_{1y}\hat{\mathbf{e}}_{2y} + \hat{\mathbf{e}}_{1z}\hat{\mathbf{e}}_{2z}}{\|\hat{\mathbf{e}}_1\|} \tag{5.3}$$

and s is a scaling parameter to normalize the result to length 1.

The third eigenvector is computed as $\hat{\mathbf{e}}'_3 = \hat{\mathbf{e}}_1 \times \hat{\mathbf{e}}'_2$. In case the distance $\|\hat{\mathbf{e}}'_3 - \hat{\mathbf{e}}_3\|$ is smaller when taking the negative value $-\hat{\mathbf{e}}'_3$, then the sign of the third eigenvalue is changed. Figure 5.1 visualizes this process. The mean distance between $\|\hat{\mathbf{e}}'_2 - \hat{\mathbf{e}}_2\|$ for older data sets was found to be about 0.1 while $\|\hat{\mathbf{e}}'_3 - \hat{\mathbf{e}}_3\|$ usually was 0.2 For newer data sets these values were around 0.05 respectively 0.1. These errors are output when orthogonalizing the eigenfield.

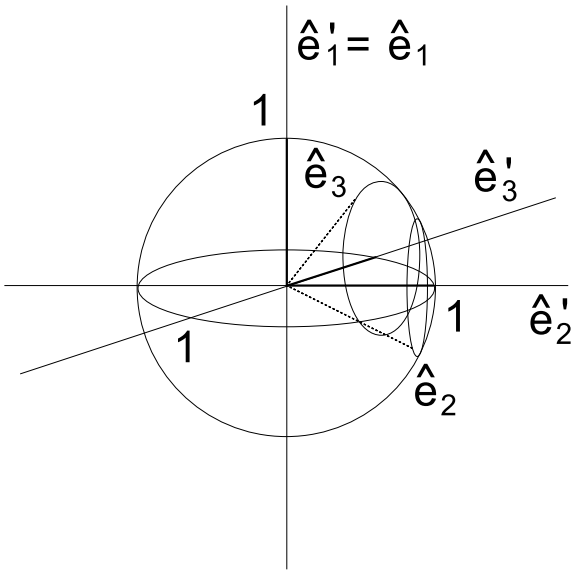

Figure 5.1: Orthogonalization of the eigenvectors

5.2.2 Filtering

Older data sets show some one-pixel wide strips in the border to the background respectively inside the brain in the border to the cerebrospinal fluid (CSF). The tensors in these strips have huge eigenvalues compared to their neighborhood. They certainly do not provide any meaningful information and should be removed completely. Therefore a simple 3 by 1 filter is incorporated in this processing step which acts like a Median filter only on locations where the left and right voxel belong to the background. No other smoothing or filtering of the data is performed at this step.

Chapter 6

Rigid Registration

Before any local comparison of data values between subjects can be done, the data sets should be rigidly aligned so that the shapes of the objects are as close to each other as possible while keeping the single shapes unchanged. This involves three basic transformations:

1. Translation
2. Rotation
3. Scaling

P. Woods [29] gives a very detailed explanation on how to apply rigid transformations. Here existing programs are used to find the transformations and apply them to the data. They are then extended to process tensor data.

The basic characteristics of rigid transformations are linearity and preservation of internal distances and angles.

6.1 Masking

First, the shape of the object of interest has to be extracted as binary data where the background is assigned the value o and the object the value $\bar{o}$. In the case of baby brains this could easily be done by thresholding the T2 weighted background image. After the threshold has been set to best extract the brain this mask can be applied to all the remaining data, i.e. the tensorfield, the eigenfield, and the other background images. The mask is stored in the standard MRI-file format for further processing (see Table 5.2 page 20).

6.2 Alignment of Grayscale Data

The task is now to find the nine transformation parameters (3 rotation angles, 3 translation values and 3 scale factors) that best align one mask with the other. Two functions are utilized. First a rough estimation is computed. This is then improved by small variations of the first approach. optimizing the result. The resulting transformation can be applied directly to all background images and the single components of the tensor field.

By applying a transformation to scalar data, the value at a voxel position (i, j, k) is displaced by a vector $U = (u, v, w)^T = (u(i, j, k), v(i, j, k), w(i, j, k))^T$. This vector will usually not be a multiple of the respective voxel size so that the original position is displaced into a position between raster points. As the resulting image has a discrete raster like the original one, the value at the destination raster position has to be estimated. Two possible estimations are implemented:

Nearest neighborhood or linear interpolation. The difference of the methods can be seen in Figures 6.2 and 6.3, where the synthetic square of Figure 6.1 was rotated by 45° degrees.

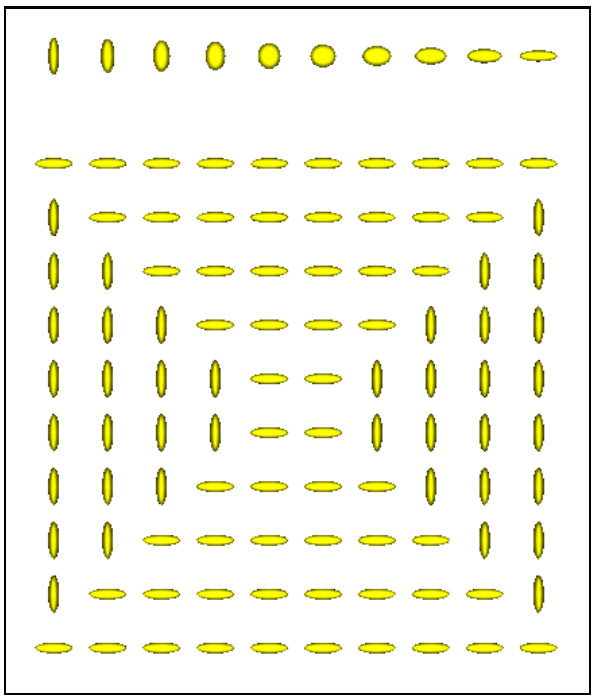

Figure 6.1: Original synthetic square

6.3 Alignment of Tensor Data

In the first step the tensors are decomposed. For each component, a scalar field is built. These scalar fields are then transformed according to the previous section and the resulting scalar fields are merged into a new, rigidly transformed tensorfield.

As tensors have an internal structure that is related to the body they belong to, any transformation has to consider local changes of this structure. That is, after merging the scalar fields, the resulting tensors have to be further processed.

For the case of a translation, this local change is trivial since the neighborhood of the tensor under inspection is displaced by the same amount. Any rotation of the body leads to a local rotation of the tensor by the same amount as the body was rotated. In the tensor-domain this leads to

$$D' = R^T D R \tag{6.1}$$

where R is the rotation matrix applied to the body and $det(R) = 1$. In the eigen-domain this is equivalent to rotating the eigenvectors by the rotation matrix:

$$\hat{\mathbf{e}}'_i = R\hat{\mathbf{e}}_i \tag{6.2}$$

It is not evident how to apply the scaling of the body locally on the tensor. Mathematically this would be

$$D' = S^T D S \tag{6.3}$$

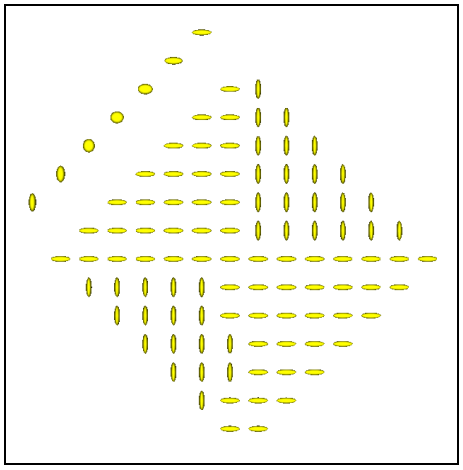

Figure 6.2: Rigid rotation of synthetic example with nearest neighbor interpolation

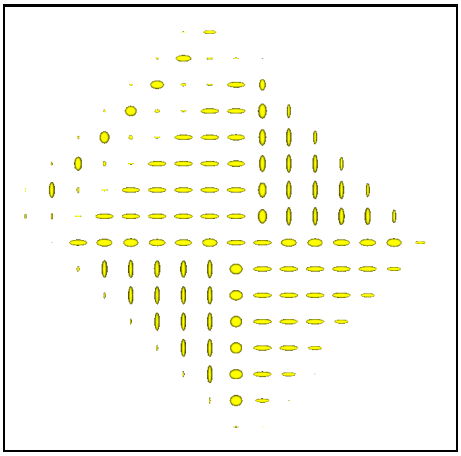

Figure 6.3: Rigid rotation of synthetic example with linear interpolation

in the tensor-domain, where S is a diagonal matrix. In the eigen-domain this would lead to a scaling of the eigenvalues

$$\lambda_i' = s_i \lambda_i \tag{6.4}$$

where s_i is a linear combination of the global scale parameters a_i according to the direction of the corresponding eigenvector $\hat{\mathbf{e}}_i$.

When considering an application like the alignment of brains, it is no longer obvious if the scaling of the tensor as a meaningful operation. Consider a small subject that is aligned to a larger one, say twice as large. Enlarging the tensors by a factor of two would mean that the diffusion in the corresponding tissue is now twice as large as before the transformation. Fibers probably do not change their diffusion properties in such a manner when the subject is growing or in different subjects. Missing values are interpolated component-wise when the rigid transformation is applied on each component, as described above, so that the consistency of the diffusion data is guaranteed otherwise. Therefore it is suggested here, that local scaling of the tensors not be applied in rigid registration. Figures 6.4 and 6.5 show the same example as in the previous section, but now with local transformation of the tensors.

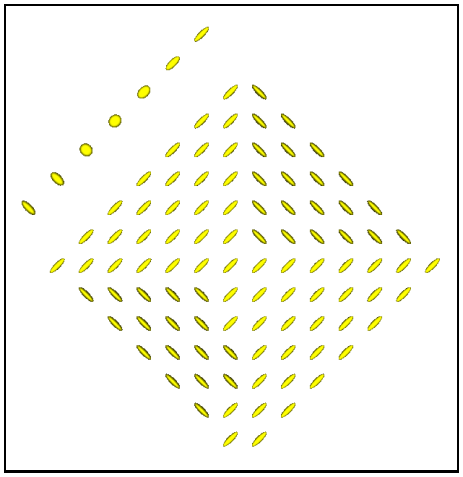

Figure 6.4: Rigid rotation of synthetic example with nearest neighbor interpolation and local transformation of tensors

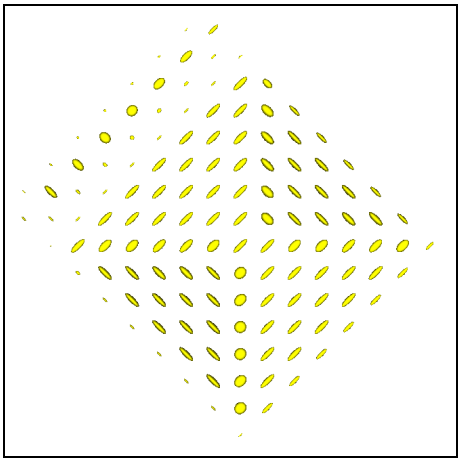

Figure 6.5: Rigid rotation of synthetic example with linear interpolation and local transformation of tensors

Chapter 7

Nonrigid Registration

Nonrigid registration is the general term for an algorithm for the alignment of data sets that are mismatched in a nonlinear or nonuniform manner. The term "matching" is used to refer to any process that determines correspondences between data sets [32].

This chapter discusses all the methods that have been used to align two tensorfields with a nonrigid registration. The following series of steps are used:

1. Extract points with high local structure in one data set.
2. For each extracted point find the best corresponding point in the second data set.
3. Check the displacement for the selected points and remove overlapping displacements.
4. Interpolate the displacement for the selected and matched points to get a displacement field for the whole data set.
5. Apply the interpolated displacement field on the second data set.
6. Eventually improve the alignment by using multiple resolutions or looping.

For simplicity in this section any field is called a three dimensional image I with values $I(i,j,k)$ at position $(i,j,k)^T$. The values $I(i,j,k)$ can therefore be any of the types scalar, tensor or eigensystem. The *s*tationary image is I_S and the *m*oving I_M. The transformation from I_M to I_S is denoted T:

$$I_S = T(I_M) \tag{7.1}$$

More precisely the goal is to find

$$I'_M = \tilde{T}(I_M) \tag{7.2}$$

which is closest to I_S:

$$I_S \simeq I'_M = \tilde{T}(I_M) \tag{7.3}$$

The displacement field is

$$U = \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} u(i',j',k') \\ v(i',j',k') \\ w(i',j',k') \end{pmatrix} \quad ,\forall \;\; i',j',k' \tag{7.4}$$

so that

$$\begin{aligned} I_S(i,j,k) &\simeq I_M(i'+u,\; j'+v,\; k'+w) \\ &\simeq I_M(i'+u(i',j',k'),\; j'+v(i',j',k'),\; k'+w(i',j',k')) \end{aligned} \tag{7.5}$$

A point P at position $(i,j,k)^T$ in I_S in the stationary image has a neighborhood $\mathcal{N}_S(P)$ which is a small window of size $\sigma_S \times \sigma_S$. A neighborhood of a point Q at position $(i',j',k')^T$ in I_M of size $\sigma_M \times \sigma_M$ is referred to as $\mathcal{N}_M(Q)$. These notation conventions are summarized in Figure 7.1. Furthermore a collection of n points $\mathcal{L} = \{P_i | P_i \in I, i = 1..n\}$ with $\mathcal{L} \subset I$ is denoted $\mathcal{L}_P(I)$.

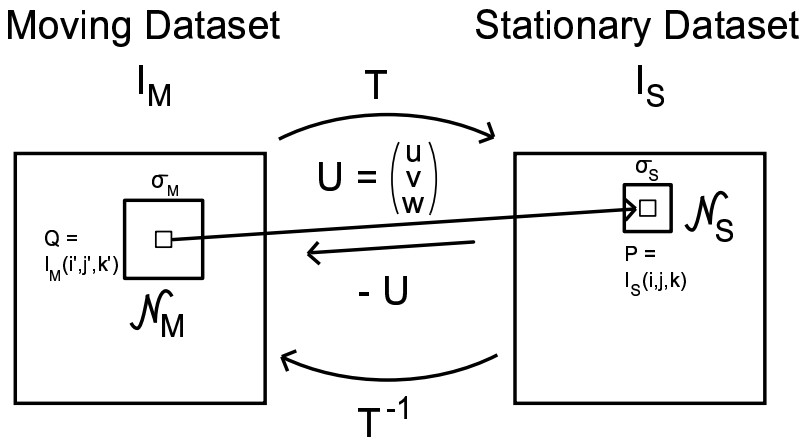

Figure 7.1: Notation conventions used

Every point Q for which $U(Q)$ is known can be displaced to a new location $P = Q + U(Q)$. If $U(Q)$ is known for every point $Q \in I_M$ the whole image I_M can be displaced. The resulting image I'_M though will have points $\bar{P}$ which have not been assigned a value from I_M. That is, the function T is not directly invertible, since U is not necessarily onto and not necessarily one to one.

As a result an image I'_M is expected that has values everywhere. Instead of interpolating the points P to get missing values for $\bar{P}$ the matching procedure can be inverted to get a function T^{-1}. This second approach is used here. Thus points are selected in the stationary image I_S. When the corresponding points have been found and the displacement field interpolated this leads to a displacement U^{-1} approximated by $-U$ from the stationary to the moving image. Now for every position in a resulting image I'_M the position where the values came from is known, and the transform is onto, so that $\mathcal{L}_{\bar{P}}(I'_M)$ is empty.

7.1 Point Extraction

To find a correspondence between two points both points must be uniquely identifiable in a certain neighborhood, so that the correspondence is unambiguous. The term "certain neighborhood" means a region that is large enough so that the corresponding point certainly lies in it but small enough so that no two equal correspondences are present. This is well known as the "Aperture Problem". This section assumes that this constraint is met.

A point on a line for example cannot be unambiguously matched to a point on a line in the second image if no additional information is provided. A corner instead has a unique counterpart in both images and can therefore be matched uniquely, of course only if the corner is present in both images. A measure of cornerness tells how much structure there is around a certain point.

In the scalar case the derivatives of the image I are computed and the outer product of the resulting vector is built, which is by definition the correlation matrix H. With the derivatives

$$I_x = \frac{\partial}{\partial x} I; \quad I_y = \frac{\partial}{\partial y} I; \quad I_z = \frac{\partial}{\partial z} I; \tag{7.6}$$

H becomes

$$H = \begin{pmatrix} I_x \\ I_y \\ I_z \end{pmatrix} (I_x \;\; I_y \;\; I_z) = \begin{pmatrix} I_x^2 & I_x I_y & I_x I_z \\ I_y I_x & I_y^2 & I_y I_z \\ I_z I_x & I_z I_y & I_z^2 \end{pmatrix} \tag{7.7}$$

which is a symmetric matrix where the determinant always equals zero. A classical measure of local structure is to use the local expectance of the values in H, i.e. averaging the components of H over a window of size $\sigma_N \times \sigma_N$.

$$\widehat{H} = \begin{pmatrix} \widehat{I_x^2} & \widehat{I_x I_y} & \widehat{I_x I_z} \\ \widehat{I_y I_x} & \widehat{I_y^2} & \widehat{I_y I_z} \\ \widehat{I_z I_x} & \widehat{I_z I_y} & \widehat{I_z^2} \end{pmatrix} \tag{7.8}$$

This matrix may have a nonzero determinant, since $\widehat{I_x^2}\widehat{I_y^2} \neq \widehat{I_y I_x}\widehat{I_x I_y}$, as long as $\widehat{I_x}, \widehat{I_y}, \widehat{I_x} \neq 0$. An eigensystem decomposition of $\hat{H}$ allows a classification of the point under inspection in edge, corner or flat region [1]. For the two dimensional case this is illustrated in Figure 7.2.

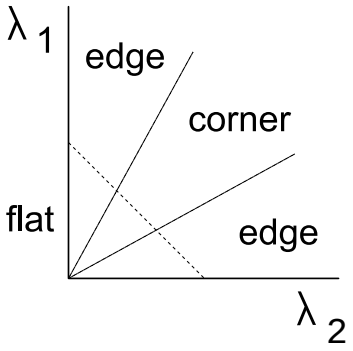

Figure 7.2: Eigensystem decomposition of the correlation matrix

$\hat{H}$ can be seen as a tensor of order 2. The eigenvalues can be illustrated by an ellipsoid. The rounder the ellipsoid the bigger the cornerness, i.e. the measure close-sphere in Equation 4.15 is a measure for cornerness in this case.

For any symmetric matrix the trace is the sum of the eigenvalues

$$trace(H) = \sum_i \lambda_i(H) \tag{7.9}$$

as can be seen by using Equation 4.8. The determinant is the product of the eigenvalues

$$det(H) = \prod_i \lambda_i(H) \tag{7.10}$$

Using only these measurements to obtain a value for the cornerness avoids doing an eigensystem decomposition.

Ruiz and co-workers [15] propose to use points above a threshold

$$t_1 = \frac{det(\widehat{H})}{trace(\widehat{H})} \tag{7.11}$$

and remove false detections by thresholding the result with

$$t_2 = \frac{det(\widehat{H})}{trace(\frac{1}{N}\widehat{H})^N} \tag{7.12}$$

with N the dimension of the image, i.e. usually 2 or 3. The value of t_2 varies between one and zero depending on the shape of the ellipsoid, one being a perfect sphere.

Here a modified version is used, where the two thresholds are incorporated in one formula. The structure S is then defined as

$$S = \frac{N \cdot det(\widehat{H})^N}{trace(\widehat{H}) + \sigma\, max(trace(\widehat{H}))} \tag{7.13}$$

where σ can usually be set to 1% or 0.01. S will have values between zero and close to one.

Using the expectance causes blurring of the point-selection since high values are diffused. The trace of H has a non-zero value without averaging and is an edge detector [1], as

$$trace(H) = I_x^2 + I_y^2 + I_z^2 \tag{7.14}$$

There are now several options to improve the point-selection. Before the process is started an anisotropic filter can be applied to enhance edges. After using a point-selection as described the result can be masked with the trace of H (not $\hat{H}$!) to undo the blurring effect of the expectance computation. Finally the local maxima of a region of the function 7.13 can be selected to get single voxel positions.

In the case of tensorfields the points should be extracted from a structure with six independent values. For each independent component of the tensor a matrix $\widehat{H_i}$ can be computed and the sum over the components of these matrix used as a final matrix $\hat{H}$

$$\widehat{H} = \sum_{i=1}^{6} \widehat{H_i} = \begin{pmatrix} \sum \widehat{h_{ixx}} & \sum \widehat{h_{ixy}} & \sum \widehat{h_{ixz}} \\ \sum \widehat{h_{iyx}} & \sum \widehat{h_{iyy}} & \sum \widehat{h_{iyz}} \\ \sum \widehat{h_{izx}} & \sum \widehat{h_{izy}} & \sum \widehat{h_{izz}} \end{pmatrix} \tag{7.15}$$

Expectance and sum are commutative, i.e.

$$\widehat{H} = \sum_{i=1}^{6} \widehat{H_i} = \widehat{\sum_{i=1}^{6} H_i} \tag{7.16}$$

The collection of points that have enough structure and have therefore been selected, is referred to as $\mathcal{M}(I_S)$

The parameters for the point-selection in the program `nonrigidreg` are described in Table A.3 in page 61.

7.2 Matching

The matching process involves two steps: First finding the best corresponding points in the second image and second locally optimizing the displacements found.

For each selected point P in I_S a neighborhood $\mathcal{N}_S(P)$ is selected. At the same position in the moving image I_M a search window $\mathcal{N}_M(P)$ is selected, i.e. an area where the corresponding point is assumed to lie. For each point $Q \in \mathcal{N}_M(P)$ in this search-window a neighborhood $\mathcal{N}_S(Q)$ is compared to the neighborhood $\mathcal{N}_S(P)$ and a value measuring the result of the comparison assigned to Q. Two different measurements of similarity for scalar data are implemented.

Maximal normalized cross correlation The normalized cross correlation (NCC) between two windows $\mathcal{N}_S(P)$ and $\mathcal{N}_S(Q)$ of same size $\sigma_S \times \sigma_S$ is defined as

$$\mathrm{NCC}(P,Q) = \frac{\sum_{k \in \mathcal{N}_S} I_S(k) \cdot I_M(\sigma_S^2 - k)}{\sqrt{\sum_{k \in \mathcal{N}_S} I_S^2(k) \cdot \sum_{k \in \mathcal{N}_S} I_M^2(\sigma_S^2 - k)}} \tag{7.17}$$

This is computed for every position $Q \in \mathcal{N}_M$. The position where the NCC is maximal, i.e. closest to one, is the best match and therefore the selected correspondence.

Least square error (LSE) Here the distance between the image values is used as measure

$$\mathrm{LSE}(P,Q) = \sum_{k \in \mathcal{N}_S} \|I_S(k) - I_M(k)\| \tag{7.18}$$

and the minimal value for all $Q \in \mathcal{N}_M(P)$ is the location where the matching point is selected.

Search Strategy In both cases, the search strategy is based on brute force, that is, no local optimization to find the best match is done. The windows that are compared, $\mathcal{N}_S(P)$ and $\mathcal{N}_S(Q)$, can be weighted with a Gaussian function prior to comparison. This makes the matching less sensitive to the size of the window and increases the "importance" of the center points P and Q.

Instead of simply selecting the best matching value the resulting vectors $\mathrm{NCC}(P,Q)$ respectively $\mathrm{LSE}(P,Q)$ are sorted so that $\mathrm{NCC}(P,0)$, $\mathrm{LSE}(P,0)$ are the best and $\mathrm{NCC}(P,\sigma_S^2)$, $\mathrm{LSE}(P,\sigma_S^2)$ the worst matches. Before accepting a match the sorted vectors can be analyzed. If the best and worst match are too close to each other, i.e. $\mathrm{NCC}(P,0) < 2{\cdot}\mathrm{NCC}(P,\sigma_S^2)$, there is not enough structure in the search-window $\mathcal{N}_M(P)$ so that the match should be ignored. Furthermore if there are several equally or almost equally good matches the displacement field in the neighborhood should be considered and matching should additionally be based on smoothness of the over-all displacement field. Only the first approach has been implemented.

In a second step, when for all points $P \in \mathcal{M}(I_S)$ a correspondence has been found, the displacements are checked. Overlapping displacements, i.e. displacement vectors that cross each other are eliminated since the tissue of a subject should not be folded. Whenever two displacements cross each other the shorter displacement is kept and the larger removed.

7.3 Kriging

In order to obtain a displacement field for the whole volume, the sparse displacements obtained at single locations have to be interpolated. This section describes the selected interpolation method. It is assumed that the sparse displacement field has an unknown underlying random field. References [35] and [38] give a good introduction and a simple example on how to use Kriging.

Kriging interpolation originates from geostatistics and is known to be the best linear unbiased estimator because it is theoretically capable of minimizing the estimation error variance while being a completely unbiased estimation procedure [34].

Kriging is a modified linear regression technique that estimates a value at a point by assuming that the value is spatially related to the known values in a neighborhood near that point. Kriging computes the value for the unknown data point using a weighted linear sum of known data values. The weights are chosen to minimize the estimation error variance and to maintain unbiasedness in the sampling. Unlike other techniques for scalar values, Kriging bases its estimates upon a dynamic, not static, neighborhood point configuration and treats those points as regionalized variables instead of random variables. Regionalized variables assume the existence of regions of influence in the data. In Kriging each region is analyzed to determine the correlation or interdependence among the data in the region and this is encoded through a function called a variogram. For the unknown value $\hat{Z}$ at the position p within the neighborhood of known points P_i with known values $Z_i(P_i)$ the basic Kriging equation is:

$$\hat{Z}(P) = \sum_{i=1}^{n} w_i Z_i(P_i) \tag{7.19}$$

Z is the actual value at a point p, and n is the number of known points used to compute $\hat{Z}$. The Z_i's are the regionalized variables and the w_i's the weights. Unlike other techniques which also use a weighted sums, in Kriging the weights are not selected based solely upon the distance between sampled and unsampled points. Kriging does not assume that the variability of the data is linear [33].

Optimal weights are determined by enforcing the error expectation in the estimate be zero

$$E(\hat{Z} - Z) = 0 \tag{7.20}$$

and the error variance be minimal

$$VAR(\hat{Z} - Z)^2 = minimal \tag{7.21}$$

where E is the expected value or mean and $VAR(\hat{Z} - Z)^2$ the mean-square-error of the dissimilarity between the two variables $\hat{Z}$ and Z. These two conditions make $\hat{Z}$ the best linear unbiased estimator and are the base equations to derive the Kriging system of equations. Furthermore the unbiasedness implies that the weights must sum up to one:

$$\sum_{i=1}^{n} w_i = 1 \tag{7.22}$$

As an exact interpolator, Kriging predicts known values with zero error. Using the method of Lagrange Multipliers it is possible to obtain a linear equation for the weights w_i of the estimator, where $\gamma(\| P_i - P_j \|)$ is the evaluation of the variogram between the points P_i and P_j:

$$\begin{pmatrix} \gamma(0) & \gamma(\| P_2 - P_1 \|) & \cdots & \gamma(\| P_n - P_1 \|) & 1 \\ \gamma(\| P_2 - P_1 \|) & \gamma(0) & \cdots & \gamma(\| P_n - P_2 \|) & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \gamma(\| P_n - P_1 \|) & \gamma(\| P_n - P_2 \|) & \cdots & \gamma(0) & 1 \\ 1 & 1 & \cdots & 1 & 0 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \\ \mu \end{pmatrix} = \begin{pmatrix} \gamma(\| P_0 - P_1 \|) \\ \gamma(\| P_0 - P_2 \|) \\ \vdots \\ \gamma(\| P_0 - P_n \|) \\ 1 \end{pmatrix} \tag{7.23}$$

Variogram Models The variogram expresses the variability of a spatial process as a function of distance and direction. Suppose the data is collected according to a random spatial process in $\mathbb{R}^n$, i.e. we observe $Y(P_i), i \in \mathbb{R}^n$. It is assumed that $VAR(Y(P_i) - Y(P_j))$ is only a function of the distance vector $P_i - P_j$. The process is said to be isotropic if it only depends upon $\| P_i - P_j \|$, the Euclidean distance between sites. Then the function $2\gamma(\| P_i - P_j \|) = VAR(Y(P_i) - Y(P_j))$ is called the *variogram*, while $\gamma(\| P_i - P_j \|)$ is the *semivariogram*. Typically, the variogram is assumed to increase with the distance $d = \| P_i - P_j \|$, assuming that the difference between pairs of observations closer in space should tend to exhibit less variability than that for pairs further apart. It is often the case that after a certain distance, called the *range* a, γ will level out at a value called the *sill*. The range is therefore the distance beyond which the deviation in the values does not depend on distance and hence values are no longer correlated.

In addition, there may be a discontinuity at the origin, the so-called *nugget* effect. This means that the fitted model does not pass through the origin, but intersects the y-axis at a positive value of $\gamma(0)$, which is τ^2. This quantity is an estimate of E, the residual, that represents spatially uncorrelated noise associated with any value of a random variable Z at P. This terminology is illustrated in Figure 7.3.

The relationship between the variogram and the covariance is given by [36]

$$2\gamma(d, \tau^2, \sigma^2, a) = 2(\tau^2 + \sigma^2(1 - \rho(d, a))) \tag{7.24}$$

where the sill is $\tau^2 + \sigma^2$, the range a and $\rho(d, a)$ a parametric correlation function. The semivariogram is also a graphical display of γ, i.e. semivariance, versus distance.

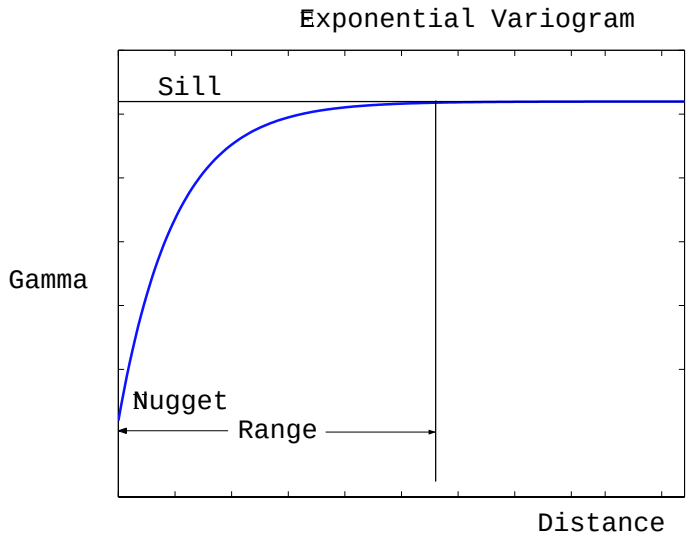

Figure 7.3: Variogram parameters

Name	$1-\rho(d,a)$
Linear	$\frac{d}{a}$
Exponential	$1-e^{-\frac{d}{a}}$
Gaussian	$1-e^{-(\frac{d}{a})^2}$
Spherical	$\begin{cases} 0.5(\frac{d}{a})^3-1.5\frac{d}{a} & \text{if } d\le a \\ 0 & \text{else} \end{cases}$
Cubic	$\begin{cases} 7(\frac{d}{a})^2-8.75(\frac{d}{a})^3+3.5(\frac{d}{a})^5-0.25(\frac{d}{a})^7 & \text{if } d\le a \\ 0 & \text{else} \end{cases}$

Table 7.1: Common Parametric Correlation Forms ([36] and [35]).

Table 7.1 is a collection of different parametric correlation functions. Figure 7.4 shows the semivariogram (Equation 7.24) for the functions in Table 7.1 with $\tau^2 = 0$ and $\sigma^2 = 1$.

In most cases the variogram is unknown and is approximated by a process called structural analysis. As there is no prior information on the resulting displacement field this approach is not applicable. Instead different variogram models are implemented so that the best model can be empirically determined by comparing the results of the registration process.

An example interpolation is shown where a synthetic example is randomly sampled at 10% and interpolated using the linear variogram model and two different neighborhoods (see Figure 7.5).

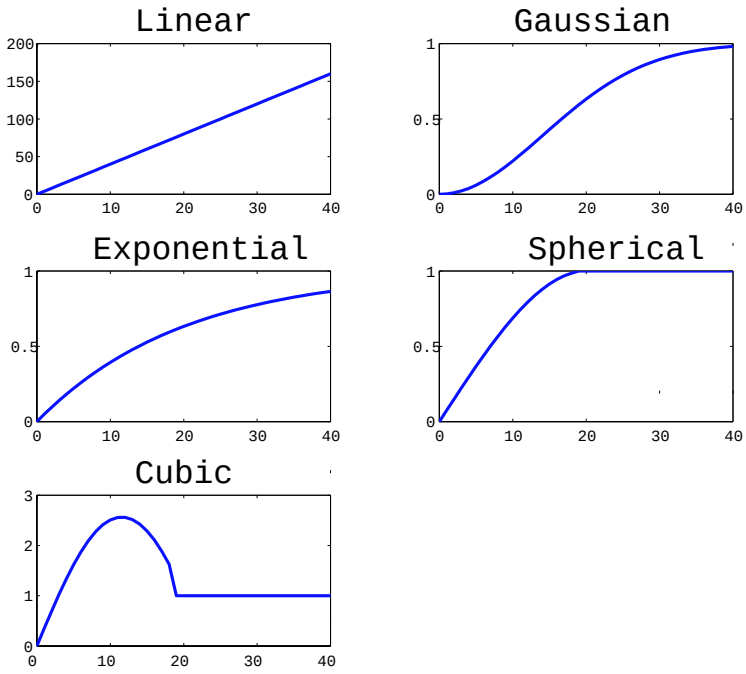

Figure 7.4: Plot of common parametric correlation forms $1 - \rho(d, a)$ used for Kriging interpolation

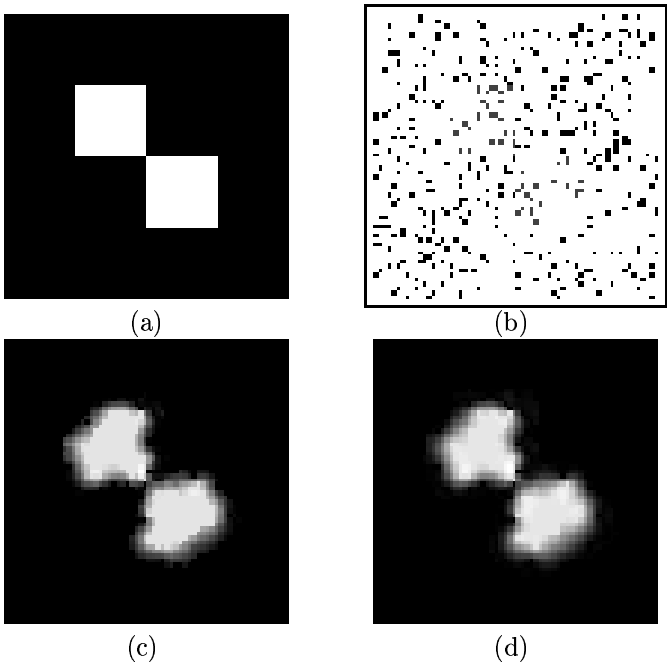

Figure 7.5: Example of using Kriging interpolation. (a) Original image; (b) Randomly selected 10% of the points; (c) Interpolation using linear Kriging and neighborhood 5; (d) Interpolation using linear Kriging and neighborhood 10. It can be seen that in (d) a value reaches further so that the transition from black to white is smoother.

7.4 Local Warping of Tensors

After the displacement field is known for every position in the image I_S the image $I'_M = T(I_M)$ can easily be computed since every location in I'_M "knows" where it comes from. This is done for each component separately. Again, as with the rigid registration, the problem arises, that tensors are structures and have to be locally transformed according to the displacement.

Three different local transformations are presented and discussed here. The first is proposed in reference [15]. D' is the tensor in image I_M that is displaced by U to a new position P. The transformation is T. Then the local transformation is applied on D' to get the final tensor D at the position P.

Local Warp with Scaling The deformation gradient A is computed which is the differential of the transformation or the Jacobian matrix of the mapping

$$A = \begin{pmatrix} \frac{\partial}{\partial x}T_x & \frac{\partial}{\partial y}T_x & \frac{\partial}{\partial z}T_x \\ \frac{\partial}{\partial x}T_y & \frac{\partial}{\partial y}T_y & \frac{\partial}{\partial z}T_y \\ \frac{\partial}{\partial x}T_z & \frac{\partial}{\partial y}T_z & \frac{\partial}{\partial z}T_z \end{pmatrix} \tag{7.25}$$

where T_i is the transformation in the ith dimensions. Combining Equations 7.1 and 7.6

$$T(I_M) = I_M(i' + u(i', j', k'),\ j' + v(i', j', k'),\ k' + w(i', j', k')) \tag{7.26}$$

A can be expressed in terms of the displacement $U = (u, v, w)^T$

$$\begin{aligned} A &= \begin{pmatrix} \frac{\partial(x+u)}{\partial x} & \frac{\partial(x+u)}{\partial y} & \frac{\partial(x+u)}{\partial z} \\ \frac{\partial(y+v)}{\partial x} & \frac{\partial(y+v)}{\partial y} & \frac{\partial(y+v)}{\partial z} \\ \frac{\partial(z+w)}{\partial x} & \frac{\partial(z+w)}{\partial y} & \frac{\partial(z+w)}{\partial z} \end{pmatrix} \\ &= \begin{pmatrix} 1 + \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} & \frac{\partial u}{\partial z} \\ \frac{\partial v}{\partial x} & 1 + \frac{\partial v}{\partial y} & \frac{\partial v}{\partial z} \\ \frac{\partial w}{\partial x} & \frac{\partial w}{\partial y} & 1 + \frac{\partial w}{\partial z} \end{pmatrix} \end{aligned} \tag{7.27}$$

The local relation of D and D' is then

$$D = A^T D' A \tag{7.28}$$

This local mapping of a tensor includes rotation, scaling and distortion of the tensor. Figure 7.7 illustrates the result of this method on a synthetic square (see Figure 6.1 on page 23). The applied displacement field is shown in Figure 7.6.

As can be seen, the shape of some tensors has been changed quite a bit.

Local Warp without Scaling In a second approach the scaling of the tensors is removed while still allowing the tensor to change shape. This is done by rescaling the result of Equation 7.28 so that the determinant of D is the same as for D':

$$D = \frac{1}{det(A)^{2/3}} A^T D' A \tag{7.29}$$

The result is displayed in Figure 7.8.

Local Rotation Using the Single Value Decomposition (SVD) the matrix A can be decomposed in a pure rotation component R and a strain component W. The SVD for any non-singular square matrix is given by

$$A = U \Sigma V^T \tag{7.30}$$

where U and V are orthogonal matrices and Σ is a diagonal matrix. The matrix A can be now written in the form

$$A = WR \tag{7.31}$$

where $W = UV^T$ is a matrix with orthogonal columns and $R = V\Sigma V^T$ is a symmetric, positive semidefinite matrix [1]. A is said to be a *pure strain* if $W = I$ with I the identity matrix, while if $R = I$, A is called a *rigid rotation* at this position [15]. Therefore any deformation of the tensor can be avoided by applying W in Equation 7.28 instead of A. For the case of the synthetic square this can be seen in Figure 7.9. This form of local rotation is the transformation mentioned in Section 4.2 and shown in Figure 4.6.

Again, as was done for the rigid transformation, it is argued here to use this transformation when nonrigidly aligning the tensor images. When aligning two different subjects the structures should not be changed. In the scalar case gray-values are displaced to the new position but their value is not changed. Equivalently in the case of tensors, a local rotation is part of the transformation but changing the shape is not obvious and meaningful in all cases. The meaning of any local change would have to be studied for each application separately and even separately for each tissue, so that it is safest to not apply any strain. Also, if the point-extraction and the matching part of the nonrigid registration are based on measurements of the tensors that depend on the shape, changing the shape after the transformation leads to a change of the similarity. This can make the chosen displacement appear wrong after the local transformation.

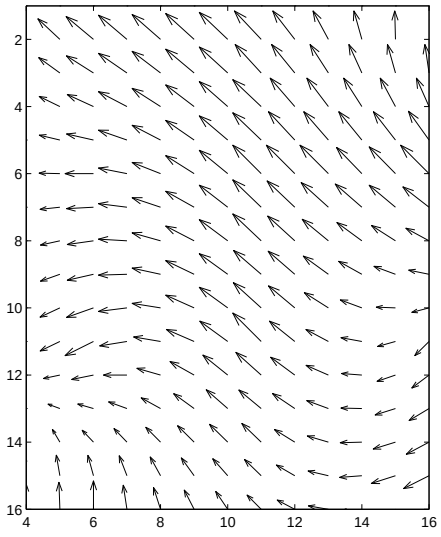

Figure 7.6: Synthetic displacement applied, used to visualize local warping

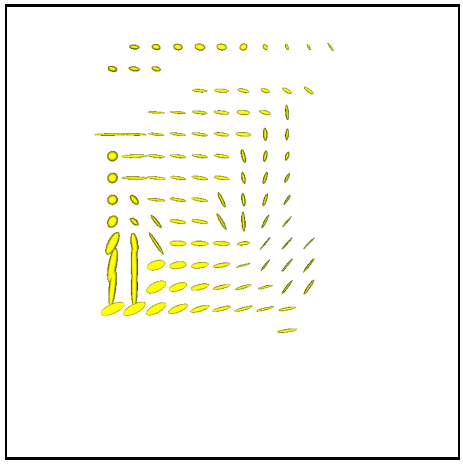

Figure 7.7: Distorted synthetic square with full local warping

The implementation allows to choose between any of the described local transformations (see Table A.3).

7.5 Multi-scale Matching

The overall process can be improved in two ways; looping and matching using multiple resolutions. Looping is simple since the second loop does not need to know anything about the previous processing and can be seen as a completely independent matching process. If the overall displacement

[1] http://www.cs.ut.ee/~toomas_l/linalg/lin2/node26.html

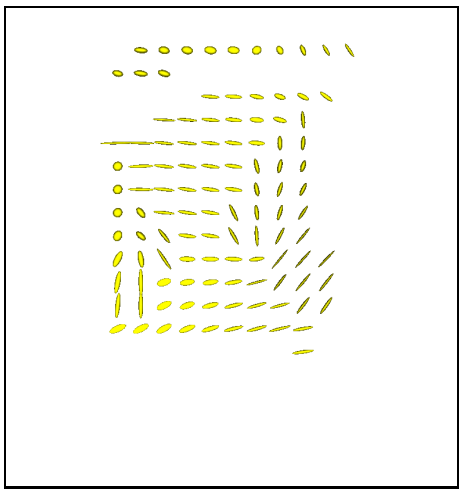

Figure 7.8: Distorted synthetic square with local warping but without scaling

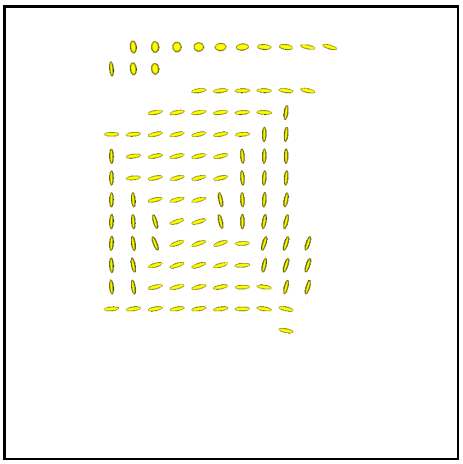

Figure 7.9: Distorted synthetic square only with local rotation of the tensors

field should be known, i.e. not only the final match is of interest, the problem of combining the displacement fields is the same as for multiple resolution matching and will be discussed there. Figure 7.10 illustrates the principle of multi-scale matching as it has been implemented. The single steps are:

1. Optionally smooth the given images I_M and I_S with a Gaussian filter. This step is not useful when working with binary or segmented data, but for all data types that have continuous values.
2. Downsample the image to get $I'_M, I''_M ... I^n_M$ and $I'_S, I''_S ... I^n_S$
3. a) Select points in the lowest level I^n_S
   b) Threshold the selection
4. Find the matches for the selected points, i.e. the displacements at this positions.
5. Upsample the displacements to all the higher levels.
6. Interpolate the displacements
7. Apply the interpolated displacements to the corresponding image I^i_M
8. Copy the resulting image back
9. While $n > 0$ set $n = n - 1$, else stop
10. Go to step 3

This form of multi-scale matching can certainly be improved. E.g. step 7 the displacement in higher levels would not need to be applied. Instead the resulting displacements could be added and in a final step the overall displacement could be applied on I_M. Combining two displacement fields though is not a trivial problem. For at least one of the transformations the inverse has to be known, but T^{-1} cannot be computed directly as has been explained in the beginning of this chapter.

The size of the search-window is adapted to the current scale. For the lowest scale it is the specified search-window, divided by the number of scales times the scale-factor. When moving

towards higher levels the search-window is a bit larger than twice the scale-factor, since any larger displacement should have been found in the lower levels.

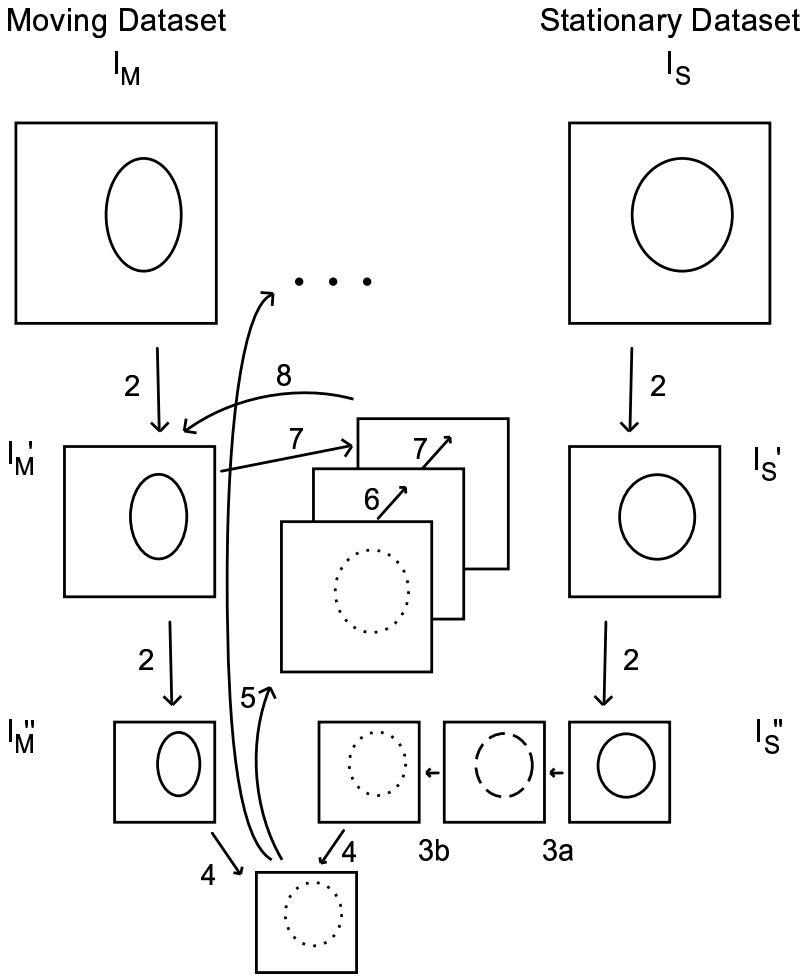

Figure 7.10: Multi-scale matching. Explanation see Section 7.5.

7.6 Measuring Results

To test the nonrigid registration synthetic displacement fields were generated and applied on different test images. The synthetic displacement U_{synth} is generated separately and independently for each component u, v, w. First a maximal displacement d_{max} is fixed. Then a regular grid with distance $d_{grid} \geq d_{max}$ is built and each position is assigned a random value x between $-\frac{d_{max}}{2} \leq x \leq \frac{d_{max}}{2}$. The sparse grid is interpolated using Kriging with a linear variogram model so that smooth random displacement fields are generated. U_{synth} is then applied to I_S to get a test image I_M. Then the nonrigid registration technique described in the previous sections is used to align I_M back to I_S.

Measuring the results, i.e. the improvement from

$$\int_{\Omega} similarity(I_S, I_M) d\Omega \tag{7.32}$$

to

$$\int_{\Omega} similarity(I_S, T(I_M))d\Omega \tag{7.33}$$

where Ω is the data set, clearly depends on the similarity measurement. A visual inspection of the image $I_S - T(I_M)$ gives a very good impression of the results.

For the binary test data measuring the improvement is trivial and boils down to be the number of voxel positions that have different values in each image. The same principle can be used when matching segmented data. Here the number of positions with different classes in each image is a meaningful measure for dissimilarity. In grayscale data counting positions where the graylevels are different is not very suggestive since this will be the case almost everywhere in the body. Of course the number should decrease as the border of the body should be better aligned after the registration, but this will be a small percentage. The total distance

$$\sum_{i,j,k} \|I_S(i,j,k) - T(I_M(i,j,k))\| \tag{7.34}$$

is much more suitable since it linearly weights the difference at each position. Similarity between tensor data sets needs to be defined to be able to use the last formula. Reference [16] proposes the inner product of two tensors

$$\sum_{i,j} D_1(i,j)D_2(i,j) \tag{7.35}$$

being the equivalent to the vector product as similarity measure.

When registering medical data the goal is to align the structures of one data set to the other data set. Ideally the structures in both data sets I_S and I_M are known, e.g. by segmenting the data. The displaced structures can then be compared.

Different tests are documented in Table 7.2. The parameters used are explained in Table A.3 in Appendix A.6. The process for the case of the chessboard is illustrated in Figure 7.11 and for the segmented baby brain in Figure 7.12.

Test images	Parameters used	Results
Chessboard and synthetic displacement	`./nonrigidreg 0 -m=0 -sw=21 -mw=9 -d=15 -r=20 -pm=2 -km=2`	1938 points different before matching, 50 after matching
MRI image and synthetic displacement	`./nonrigidreg 1 -s=pointmatcherdata/cm.001 -o=pointmatcherdata/cm -syn=pointmatcherdata/cm -mw=9 -sw=15 -m=1 -r=15`	133016 total graylevel distance before matching, 42167 after matching

Table 7.2: Comparison of results when testing with synthetic deformation fields

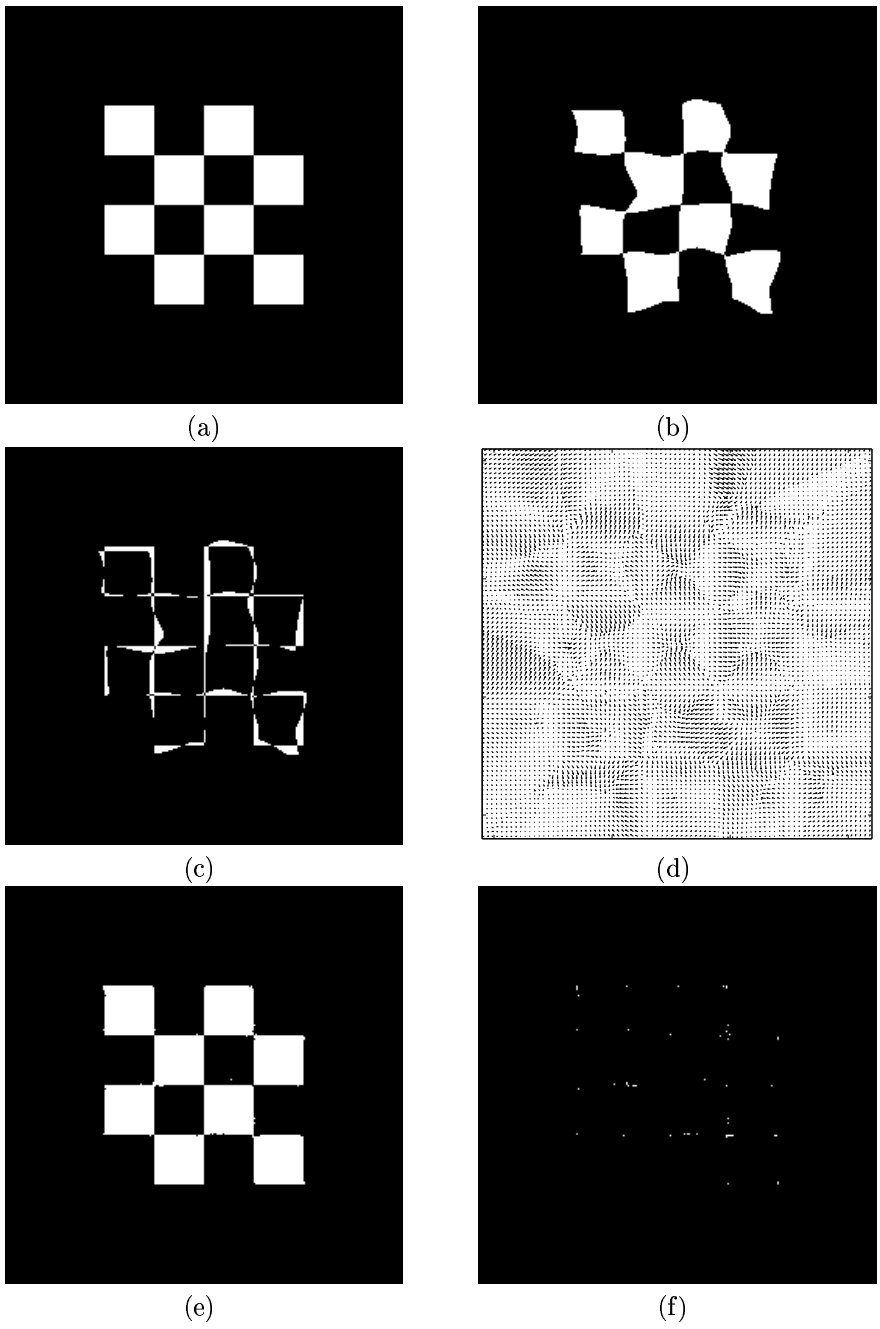

Figure 7.11: Chessboard example for the nonrigid matching process. (a) Original image; (b) Synthetically displaced version; (c) Difference between (a) and (b); (d) Displacement field for nonrigid registration from (b) to (a), zoom into the center region; (e) Resulting match; (f) Difference between (a) and (e).

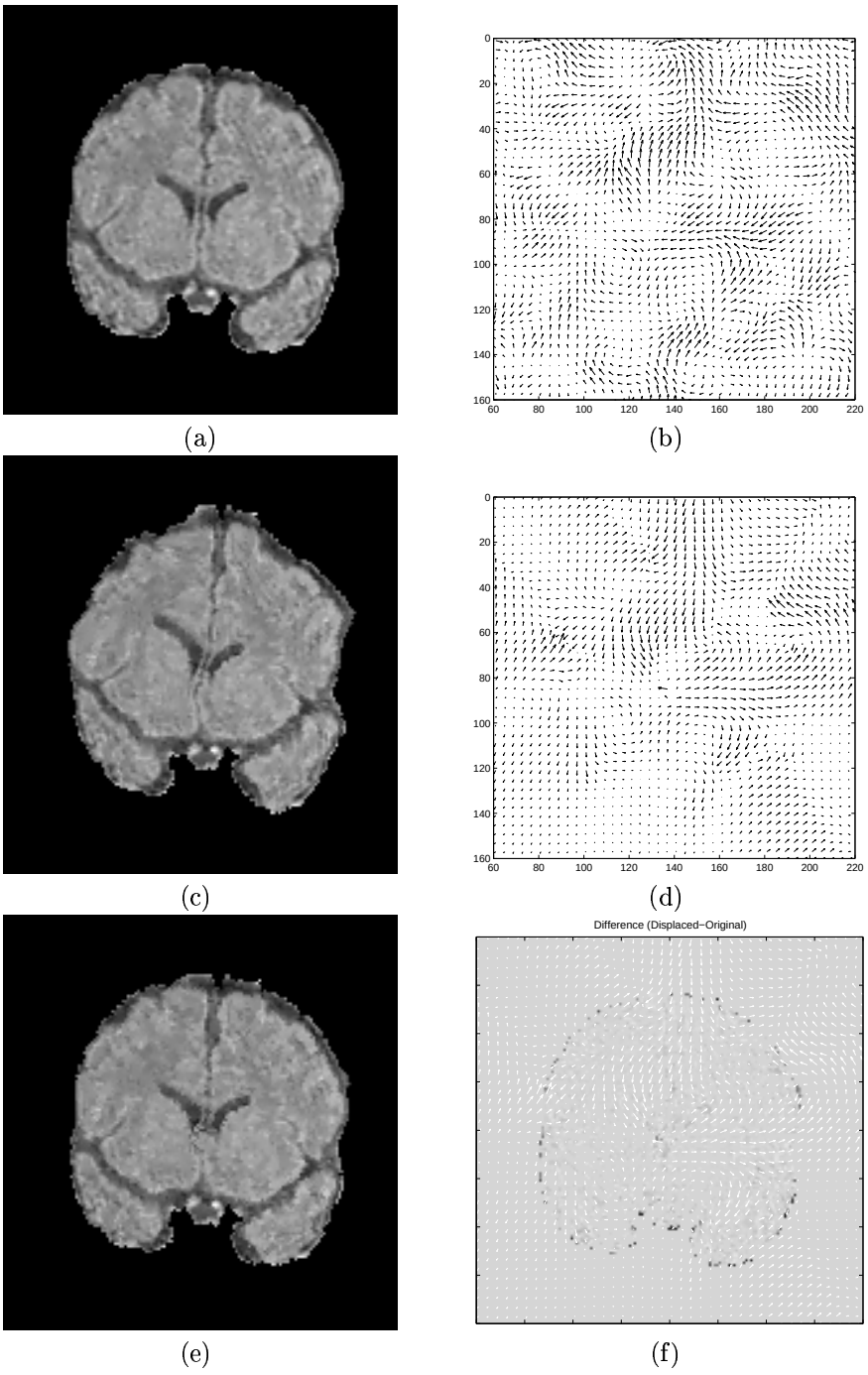

Figure 7.12: Alignment of syntheticaly displaced MRI scan to original scan as an example for the nonrigid matching process. (a) Original image; (b) Synthetically displaced version; (c) Difference between (a) and (b); (d) Displacement field for nonrigid registration from (b) to (a); (e) Resulting match; (f) Difference between (a) and (e).

Chapter 8

Classification

Classification and segmentation can be defined as follows [37]:

- *Classification* involves labeling pixels in terms of different classes by grouping pixels that have similar characteristics, based on the measurement or estimation of different features known to exist in the image using low-level operators with small areas of support. It does not demand spatially contiguous voxels within any single class.
- *Segmentation* is the parcellation of the input image into meaningful contiguous groups of voxels. The word "meaningful" indicates a task-dependent definition. This can involve the identification and delineation of commonly-recognized structures in the human brain that are labeled as such with neuroanatomy textbooks.

The classification of tensor data should connect regions with similar local structure, i.e. fibertracts in the corpus callosum. These fibertracts should be present in all subjects. An alignment of tensor data should match the corpus callosum of two subjects. As explained in the previous section the classification would also allow to test the alignment process by comparing the classified data sets.

One possible way of classification of tensor data has already been presented in Section 4.3 where the tensors have been classified depending on their shape.

This is the simplest approach where the classification is based only on the values of the voxel itself. The classification and the measurements are best based on the eigenfield, since the eigendomain has a much more intuitive meaning. If a useful classification has been found, it is worthwhile to find a corresponding one in the tensor-domain, since this would save the eigensystem decomposition of the tensorfield.

Different measurements can lead to a number of different classifications. Examples are:

- Similar shape: $\frac{\lambda_1}{\lambda_2} \simeq \frac{\lambda'_1}{\lambda'_2}$ and $\frac{\lambda_1}{\lambda_3} \simeq \frac{\lambda'_1}{\lambda'_3}$
- Similar size: $\prod_i \lambda_i \simeq \prod_i \lambda'_i$
- Similar direction of the largest eigenvalue: $\cos(\hat{\mathbf{e}_1}, \hat{\mathbf{e}'_1})$ <threshold

In the implementation a threshold t_1 is set to select certain voxel positions where the tensor is above this threshold. The image is searched from top left to bottom right for tensors above the threshold. The measurement close-line is used to select these starting points. As soon as a position is found where the tensor is above t_1, a new label is assigned to this position and the tensor compared with its neighbors. If the similarity is close enough, i.e. above a threshold t_2 then the neighbor is assigned the same label and is set to be the new location. This new location is now expanded the same way. The process stops when no neighbor is similar enough, i.e. the threshold t_2 is above

the similarity. Then the image is again searched to find the next position where the tensor has not been assigned a label yet and is above t_1.

Any tensor that is not assigned a label but is not equal to zero is assigned a "rest-group" label so that the body does not get mixed with the image background. Finally a histogram of the labeled image is build and the labels are sorted according to their size. The resulting labels can be displayed with the program `main`.

The described process is certainly in a very early stage and does not produce useful results yet. But it helps to get an insight on the tensor data and can be used as a starting point for further implementations.

The label-growing approach has also been used to "clean" the mask generated in Section 6.1. The thresholded T2 weighted image is labeled, (with the threshold t_2 for the neighborhood set to zero), so that any connected group is assigned the same label. Then only the largest label is kept, so that the brain, as the largest object in the image, is the only remaining label.

Segmentation into different tissue classes can be performed using `imager`[1]. This program allows to manually select example points of several images and does a segmentation of the data based on k-Nearest-Neighbor estimation.

The following Figure 8.1 shows some results obtained with the implemented classification method.

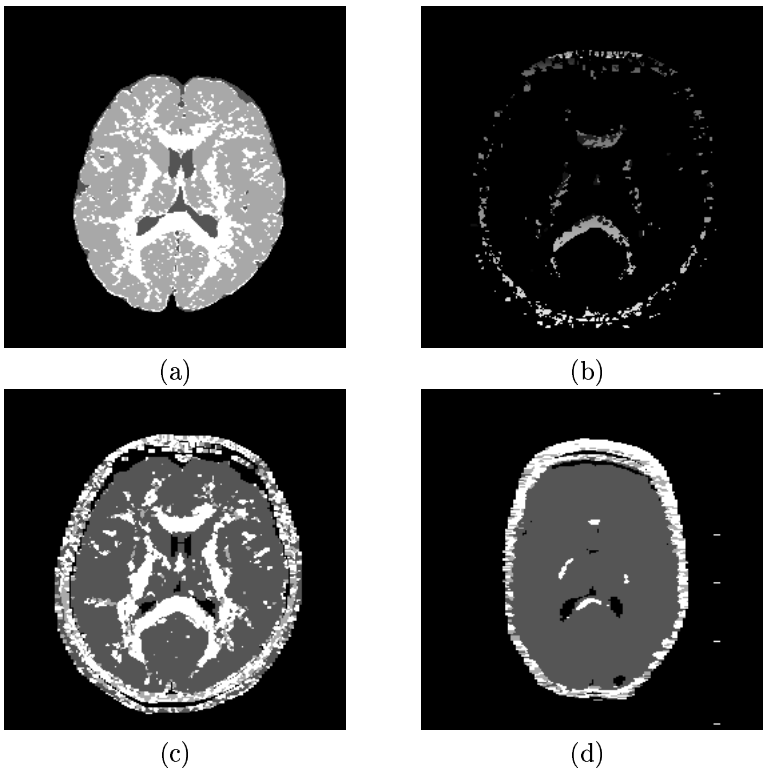

Figure 8.1: Classification of tensor data. (a) Thresholding of different measurements (T2W image, DWI image and ADC image); (b) Label growing, adult human brain, no smoothing; (c) Classification in three groups depending on the shape of the tensor, adult human brain, Smoothed with Gauss filter of length 3; (d) Same classificationa as example (c), baby brain at 40 weeks postconceptional age.

[1] written by S.Warfield

Chapter 9

Implementation

This chapter should help to get started with the program code and expand it for further needs. Appendix A is a documentation of the most important classes, programs, and the choosen xml data representation.

9.1 Class Hierarchy

Figure 9.1 shows class hierarchy. Only the main dependencies are illustrated. The libraries used are VTK for the visualization, CLAPACK for the mathematical computations and Xerces-C for processing the xml-header. The rightmost column are files containing functions that do not belong to a class and therefore do not have any private members. The bottom row shows the main applications that have been implemented. Only the main dependencies have been indicated.

tensor.h and eigen.h are classes implementing the low-level data structures for a tensor respectively an eigensystem. Every operation that acts on single elements of these types should be implemented in the respective class.

basefield.h contains the basic definitions of any field. The class itself does not contain any field though. It mainly reads and writes the xml-header. The information is stored as protected members so only derived classes can access them directly. Functions are provided to set and get the single characteristics of a data set, so that accessing these values is equal for all types of data fields. The most important informations stored in this class are the volume dimensions, the voxel dimensions and the location of the data files.

rsfunctions.h For scalar data, i.e. floating point or integer numbers, no special class has been implemented. That means that any function that works on scalar fields has no private members and therefore all information (volume and voxel dimensions) need to be provided when the functions are called.

tensorfunctions.h Functions that use the tensor data structure, that is 3×3 matrices for computations, are placed in `tensorfunctions.h`. Like functions in `rsfunctions.h` they do not belong to a class and so volume and voxel dimensions need to be provided. The most interesting functions are `localWarpTensorField(...)` for the local transformation of the tensors after they have been displaced and `findPointsInFloatArray(...)` to select the points with high local structure. `displaceTensorField(...)` applies the displacement field on every single component of a tensorfield.

The classes `tensor.h, eigen.h and basefield.h` build the bases for the current implementation and are therefore documented in Appendix A. For any class and function collection the *.h file contains a small documentation for each function.

tensorfield.h, eigenfield.h, and background.h are then all derived from `basefield.h` and inherit all its members. When initializing one of this classes the actual field of the respective type is generated, which can be empty or read from a file.

display.h contains some basic functions to display images. For scalar data the display is generated by writing a temporary `ppm` file. Then a system call opens the generated file with `xv`. The temporary files are all deleted when the destructor of the class is called.

The tensor representation as ellipsoid uses VTK. In this case the VTK renderer is used to display the image.

The remaining functions in this class can be used to display specific properties of a tensor- or eigenfield.

rsfunctions.h is not a class but a collection of functions. Any function that does not need a private member or specific data structures such as tensors or eigensystems is placed in this file.

test The program `test` is a collection of different tests that have been done to check the code. It also has been used to generate most of the examples in this report. Any test routine is implemented as a function and the main program calls the a function depending on the first parameter passed to `test`. A simple call `./test` will display the different options available.

convert, mask, rigidreg and nonrigidreg are the main programs and have been introduced in the previous chapters. By typing the programs name a list of options is displayed. The options are also explained in the respective section in Appendix A. `mask` is a simple program that takes one data set as argument. It allows to mask the data sets by thresholding the T2W image interactively.

main can be used as soon as the data has been processed with `convert`. It allows different filterings of the data and to display the data sets as described in section 4.3.

9.2 Data Representation

Volumes are represented as arrays of tensors, eigensystems or floating point numbers. The arrays are always indexed so that the x-dimension is the "fastest" growing number. A position i, j, k in a volume is therefore the position $i + j \cdot X + k \cdot X \cdot Y$ in the respective array, where X is the x-dimension and Y is the y-dimension. The classes `eigenfield.h` and `tensorfield.h` provide functions to access the positions i, j, k by specifying all three values or directly by providing the position with one value $c = i + j \cdot X + k \cdot X \cdot Y$.

Any private member begins with an underscore _. For example the tensor-structure is a 3×3 matrix and stored as a floating point precision array

```
float _t[9];
```

where the indexing is

$$\begin{pmatrix} \_t[0] & \_t[1] & \_t[2] \\ \_t[3] & \_t[4] & \_t[5] \\ \_t[6] & \_t[7] & \_t[8] \end{pmatrix}$$

The eigensystems are represented as 3 floating point numbers and 3 arrays of length 3 representing the respective eigenvectors:

```
float _lambda1, _lambda2, _lambda3;
```

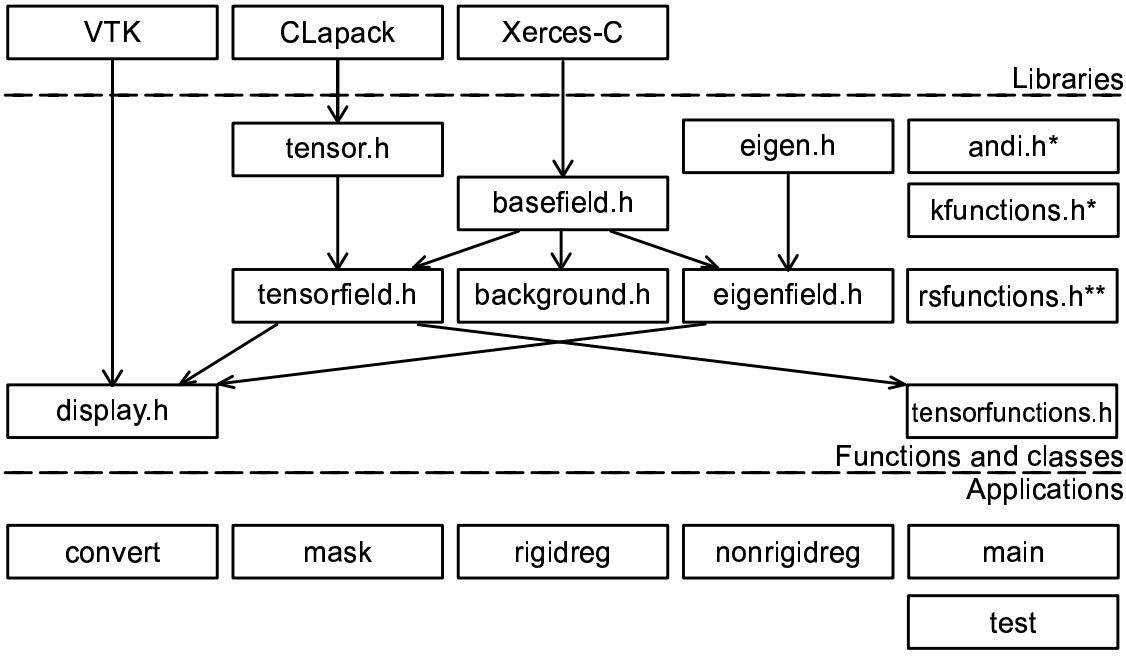

Figure 9.1: C++ Class hierarchy
* `andi.h` was written by J.Rexilius, `kfunctions.h` by S.Warfield
** includes some functions written by S.Warfield

```
float _vector1[3], _vector2[3], _vector3[3];
```
The eigensystems are sorted by the size of the eigenvector, i.e. `_lambda1` always represents the largest eigenvalue.

To store and process the data of the tensor- or eigenfield floating point precision numbers have been chosen. This seems to be the best trade-off between memory usage and precision. Problems only arose when computing the single value decomposition to generate the local rotation of the tensors in the nonrigid registration. Often, the resulting tensor was no more symmetrical so that double precision when performing the local warping is necessary.

The background images are all represented in unsigned short precision.

No additional information is stored in the data files, i.e. no header or leading bytes are present in the files. All additional information has been placed in the xml-header file, which will be described next.

Functions that take a pointer to an array of any type as parameter never check if the provided array is long enough to perform the requested operations. Boundary checks are only performed when working on the private member of a class, since the class member function knows the size of its own data field.

9.3 XML Header

To store information related to the data the xml-standard[1] has been used. This allows to collect all the information in a single, human readable file. The interaction with C++ has been done using

[1] http://www.w3.org/XML/

Xerces-C [19]. As this information is the same for all classes (background, tensor- and eigenfields) only one file needs to be stored. `basefield.h` provides the functions to read and write the xml-file in a convenient manner. Most important when adding information to the header the following steps have to be performed:

1. Change the Document Type Definition (DTD)
2. Add the new data to the protected members in `basefield.h`.
3. Add the member in the constructor and destructor.
4. Edit the read and write functions.
5. Implement a function that allows to get and set the new member.

When changing the DTD the new elements should be marked as optional, i.e. with either `?` or `*` indicating that a xml-file without this element is still a valid instance of this DTD. Otherwise all current applications will produce an error as they validate the provided xml-file with the DTD.

The functions that need to be edited in step 4 are:

```
void readdata(DOM_Node& node);
void readmanualy();
```

which are private members of `basefield.h`, and

```
void write_XMLHeader(const char xmlPath[]);
```

Eventually the function

```
void assert_values();
```

which checks if all the values are present before the header is saved into a file, also needs to be expanded.

Functions are provided to read a string, float number, and attribute values. Numbers and strings are already read and written to the xml-file, so that with simple copy-paste it should be possible to add new data.

The last step is optional, since a derived class has access to the protected members. Nevertheless it is always useful to provide such a function for direct access in an application or test program.

Chapter 10

Conclusion

The primary contributions of this thesis are:

- A complete implementation of the whole process from preprocessing of the diffusion tensor data to the nonrigid alignment of two different data sets has been implemented. The implementation also is a framework to process and store tensor data, so that further work can be based on the developed classes.
- A comparison and assessment of different tensor reorientation strategies.
- Different approaches to extract points in grayscale and tensor data are discussed and implemented.
- A new visualization method of tensor data is presented.

The new visualization method is very useful to inspect small volumes of tensor data. It is the most precise way of displaying tensor data sets as all the information in the data is represented in the ellipsoids. Displaying the tensors as a three dimensional structure is also probably the best way to understand the properties of this data structure. Furthermore, it enables a developer to prove visually the correctness of any algorithm implemented and to check if the results of any transformation of the data has a meaningful interpretation.

Besides optimizing the code for performance, further investigation of general improvements and extensions would be interesting. Some of these ideas will be presented in the next section.

10.1 Further Work

First of all, the nonrigid registration should be expanded to fully support three dimensional data. Most functions are written to work with three dimensional data but they have not been tested. The matching process itself (Section 7.2) is only implemented for two dimensional data so this would be a starting point. Finally, an extension to three dimensions involves making sure that all functions include the voxel-dimensions correctly.

Thus, the extension to three dimensions primarily means finding a method for testing the correctness of all functions.

As it can be assumed that diffusion tensor data has no meaningful interpretation in regions of gray matter, a first step in incorporating T2 weighted data would be to mask the tensor data with the segmented white matter and use the border between gray and white matter, as a boundary condition when matching diffusion tensor data.

As any displacement of tensor data can be performed in two separate steps, (displacement of the single components and local transformation of the tensor), any displacement can be applied

on tensor data. It would also be interesting to compare the results when using displacement fields derived from other then tensor data. This should be easy if the tensor data is aligned with the SPGR data. Any transformation, (rigid and nonrigid), applied on the SPGR data can also be applied on the tensor data and the corresponding local transformation of the tensors performed after the displacement of the single components.

In Section 7.3 regarding the interpolation of the data, it was mentioned that the variogram models have to be empirically selected for any given alignment. A way to find the optimal variogram model would be to select landmark points manually in a number of cases and match the corresponding points. Then the resulting displacements can be statistically analyzed to find the best variogram function for the interpolation of the displacement fields.

The visualization of the tensors as ellipsoids is only useful for small data volumes. As can be seen in the example images in Appendix B, the information provided becomes too large for an interpretation. The next step is to display the tensors as fibers, i.e. connect the tensors by following the largest eigenvalue. Fiber-tracking is a large research area; the correct identification of a path is certainly not trivial. VTK provides a class `vtkHyperStreamline` to connect tensors. As the images with the tensor-ellipsoids suggest, it should be possible, at least in certain regions with clear orientation of the tensors, to apply this VTK data structure to connect single tensors. The connected tensors are then visualized as a tube, which would represent a fiber. A prototype for such a function is included in the class `display.h`.

References

[1] Luc van Gool
Computer Vision I & II
Script for lecture in computer vision at the Swiss Federal Institute of Technology, 1999-2000

[2] Petra S. Hüppi M.D., Stephan E. Maier M.D., Sharon Peled Ph.D., Gary P. Zientara Ph.D., Patrick D. Barnes M.D., Ferenc A. Jolesz M.D., Joseph J. Volpe M.D.
Microstructural Development of Human Newborn Cerebral White Matter Assessed In Vivo by Diffusion Tensor MRI
Pediatr Res 44:584-590,, 1998.
http://splweb.bwh.harvard.edu:8000/pages/current_projects.html

[3] Petra S. Hüppi M.D., Schmucknecht T, Boesch C, Fusch C, Felbinger J, Bosse E, Herschkowitz N
Structural and neurobehavioral delay in brain development in preterm infants
Pediatr Res 39:895-901

[4] Terrie Inder M.D., Petra S. Hüppi M.D., Stephan E. Maier M.D., Ferenc A. Jolesz M.D., Don di Salvo M.D., Richard Robertson M.D., Patrick D. Barnes M.D., Joseph J. Volpe M.D.
Early detection of periventricular Leukomalacia by Diffusion-Weighted MR imaging techniques
J Pediatr, 134:631-634, 1999.
http://splweb.bwh.harvard.edu:8000/pages/current_projects.html

[5] Terrie Inder M.D.,et al.
Periventricular White Matter injury in the premature infant is followed by reduced cerebral cortical gray matter volume at term
Annals of Neurology, 46(5):755-760, 1999.
http://splweb.bwh.harvard.edu:8000/pages/current_projects.html

[6] Petra S. Hüppi M.D., Simon Warfield Ph.D., Ron Kikinis M.D., Patrick D. Barnes M.D., Gary P. Zientara Ph.D., Ferenc A. Jolesz M.D., Miles K. Tsuji M.D., Joseph J. Volpe M.D.
Quantitative Magnetic Resonance Imaging of Brain Development in Premature and Mature Newborns
Ann. Neurol. 43(2):224-235, 1998.
http://splweb.bwh.harvard.edu:8000/pages/current_projects.html

[7] Joseph J. Volpe
Neurology of the Newborn 3rd edition
WB Saunders, Philadelphia, 1995

[8] Zhi-Pei Liang, Paul C. Lauterbur
Principles of Magnetic Resonance Imaging : A Signal Processing Perspective
IEEE Press Series in Biomedical Engineering; 1999

[9] P. Boesiger
Script for lecture in Biomedical Engineering II
Abteilung Electrotechnik, ETH Zürich, Summer 2000

[10] A. Einstein.
Investigation on the theory of the Brownian movement
New York: Dover; 1926

[11] W. E. L. Grimson, R. Kikinis, F. A. Jolesz and P. M. Black
Image-guided surgery
Scientific American, p. 62-69; June 1999

[12] Todd K. Moon, Wynn C. Stirling
Mathematical Methods and Algorithms for Signal Processing
Prentice-Hall 2000

[13] Will Schroeder, Ken Martin, Bill Lorensen
The Visualization Toolkit, An Object-Oriented Approach to 3D Graphics, 2nd Edition
Prentice-Hall 1998

[14] Steve Oualline
Practical C++ Programming
O'Reilly & Associates, Inc. 1995

[15] J. Ruiz-Alzola, C.-F. Westin, S.K. Warfield, C. Alberola, S. Maier, R. Kikinis
Nonrigid Registration of 3D Tensor Medical Data
In Proceedings of Third International Conference On Medical Robotics, Imaging and Computer Assisted Surgery, Pittsburgh, Pennsylvania, USA. 2000. pp.541-550.

[16] C.-F. Westin, S.E. Maier, B. Khidhir, P. Everett, F.A. Jolesz, R. Kikinis
Image Processing for Diffusion Tensor Magnetic Resonance Imaging
1999

[17] S. Peled, H. Gudbjartsson, C-F. Westin, R. Kikinis and F.A. Jolesz
Magnetic Resonance Imaging shows Orientation and Asymmetry of White Matter Tracts
Brain Research, 780(1): 27-33, January 1998

[18] D. M. Wimberger, T. P. Roberts, A. J. Barkovich, L. M. Prayer, M.E. Moseley and J. Kucharczyk
Identification of "premyelination" by diffusion-weighted MRI
J. Comp. Assist. Tomographie 19(1):28-33, 1995

[19] *Programming Guide Xerces-C*
http://xml.apache.org/xerces-c/index.html

[20] *LAPACK – Linear Algebra PACKage*
http://www.netlib.org/lapack/

[21] A.I. Borisenko and I.E. Tarapov
Vector and Tensor Analysis with applications
Dover Publications, Inc. New York 1979

[22] Denis Le Bihan et al.
Diffusion and Perfusion Magnetic Resonance Imaging, Applications to Functional MRI
Editor: Denis Le Bihan; Raven Press 1995

[23] Denis Le Bihan
Diffusion Imaging
Section I. B. 2., Part I-IV in [22]

[24] Cooper RL, Chang DB, Young AC.
Restricted diffusion in biophysical systems
Biophys J 1974; 14:161-177.

[25] Hansen JR
Pulsed NMR study of water mobility in muscle and brain tissue
Biochim Biophys Acta 1971; 230:482-486

[26] Denis Le Bihan, Robert Turner, and Nicholas Patronas
Diffusion MR Imaging in Normal Brain and in Brain Tumors
Chapter 8, Part I in [22]

[27] Le Bihan D., Turner R., Douek P.
Is water diffusion restricted in human brain white matter?
An echo-planar NMR imaging study
Neuroreport 1993; 7:887-890

[28] H. Gudbjartsson, S.E. Maier, R.V. Mulkern, I.A. Morocz, S. Paty and F.A. Jolesz
Line scan diffusion imaging
Magn.. Reson. Med., 36:509-519, 1996

[29] Roger P. Woods
Spatial Transformation Models
CLA School of Medicine; Academic Press 2000

[30] Arthur W. Toga
Brain Warping
Academic Press, San Diego, USA; 1999

[31] John C. Mazziotta, Arthur W. Toga, Richard S.J. Frackowiak
Brain Mapping, The Disorders
Academic Press 2000

[32] Simon Warfield, Andre Robatino, Joachim Dengler, Ferenc Jolesz, Ron Kikinis
Nonlinear Registration and Template Driven Segmentation
http://splweb.bwh.harvard.edu:8000/pages/current_projects.html

[33] Stephen M. Matechik, Martin R. Stytz
Using Kriging to Interpolate MRI Data
IEEE 6/1994 P. 576-577

[34] Rob W. Parrot, Martin R. Stytz, Phillip Amburn, David Robinson
Towards Statistically Optimal Interpolation for 3-D Medical Imaging
IEEE Engineering in Medicine and Biology, Sep., p. 49-59

[35] Hans Zuuring, L. Queen
GIS: Methods and Applications I
http://www.forestry.umt.edu/academics/courses/FOR503/SPATIAL_Stats.htm#Semivar

[36] Mark D. Ecker, Alan E. Gelfand
Bayesian Variogram Modeling for an Isotropic Spatial Process
June 26, 1997

[37] D.L. Collins
3D Model-based segmentation of individual brain structures for magnetic resonance imaging data
PhD thesis, 1994

[38] J.L. Starck and F. Murtagh and A. Bijaoui
Image Processing and Data Analysis: The Multiscale Approach
Cambridge University Press, UK, 1998

Appendix A

Functions Documentation

A.1 tensor.h

Functions in this class mainly provide a way to simple use tensors and apply basic mathematical operations on them. The data stored in a tensor is a floating point precision array: `float _t[9]` where the indexing is

$$\begin{pmatrix} \_t[0] & \_t[1] & \_t[2] \\ \_t[3] & \_t[4] & \_t[5] \\ \_t[6] & \_t[7] & \_t[8] \end{pmatrix}$$

The only private function is
`void eigsort(const double *W, int *i) const;`
which sorts the eigenvalues by size and is used in `eig(...)`.

Three different constructors allow the generation of an empty tensor (all values are set to zero), a symmetric tensor where the indexing corresponds to

$$\begin{pmatrix} t1 & t4 & t5 \\ t4 & t2 & t6 \\ t5 & t6 & t3 \end{pmatrix}$$

and a full tensor with nine independent values.

```
tensor();
tensor(float t1, float t2, float t3, float t4, float t5, float t6);
tensor(float t1, float t2, float t3, float t4, float t5, float t6, float t7, float
t8, float t9);
```

Three corresponding functions are available to set the values after the initialization:

```
void setZero();
void set(float t1, float t2, float t3, float t4, float t5, float t6);
void set(float t1, float t2, float t3, float t4, float t5, float t6, float t7, float
t8, float t9);
```

`float get(const int i, const int j) const;`
returns the value at position i, j in the tensor where $0 \leq i, j \leq 2$ and
`bool isZero() const;`
checks if the tensor is empty, i.e. all values are zero and returns true if so.

The basic mathematical operations can be performed between tensors and between tensors and scalars. Dividing a scalar by a tensor is not a legal operation.

```
tensor operator+ (const tensor &s, const tensor &t);
tensor operator+ (const tensor &t, const float x);
tensor operator+ (const float x, const tensor &t);

tensor operator- (const tensor &s, const tensor &t);
tensor operator- (const tensor &t, const float x);
tensor operator- (const float x, const tensor &t);

tensor operator* (const tensor &s, const tensor &t);
tensor operator* (const tensor &t, const float x);
tensor operator* (const float x, const tensor &t);

tensor operator/ (const tensor &s, const tensor &t);
tensor operator/ (const tensor &t, const float x);

tensor operator- (const tensor &t);
```

The last functions changes the sign of a tensor.

To send a tensor to the standard output the operator
`ostream &operator << (ostream &out, const tensor &t);`
is used, which will print the tensor values like

```
Tensor values:
                              [5   20  25;
                              20   10  30;
                              25   30  15]
```

which allows easy copy-paste into matlab.

Basic mathematical operations are:

`float det() const`	the determinant
`float trace() const`	the trace (see invariant I_1 in equation 4.6)
`tensor inv(tensor &result) const`	the inverse of a 3×3 matrix
`tensor trans(tensor &result) const`	the transpose of a 3×3 matrix

To compute the eigenvalue, -vector decomposition (see equation 4.5) the CLAPACK function `dsygv_(...)` is called. The eigenvalues are returned so that they can be directly used to instantiate an eigensystem. The last argument must be 1 if the eigenvalues and eigenvectors should be sorted in descending order.

```
int eig( float &eig1, float &eig2, float &eig3, float *vector1, float *vector2, float
*vector3, const int sort=0) const;
```

The single value decomposition also calls a CLAPACK function (see equation 7.30) whereas CLAPACK returns directly V^T instead of V.

```
int svd( float &r1, float &r2, float &r3, tensor &q, tensor &s);
```

The equivalence between equation 7.30 and the function call is:

$$\begin{aligned} A &= U\Sigma V^T \\ &= \begin{pmatrix} q_{11} & q_{12} & q_{13} \\ q_{21} & q_{22} & q_{23} \\ q_{31} & q_{32} & q_{33} \end{pmatrix} \begin{pmatrix} r_1 & & \\ & r_2 & \\ & & r_3 \end{pmatrix} \begin{pmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{pmatrix} \end{aligned}$$

so that the rotation matrix $W = UV^T$ is qs.
The following measurements can be computed:

`float euclid() const`	euclidian magnitude $\sqrt{\sum_{i,j=0}^{2} D_{ij}}$
`float Max() const`	maximal value in tensor
`float Min() const`	minimum value in tensor

The boolean comparison between two tensors is based on the function
`float measure()const {return euclid();};`
so that $\leq$, $\geq$, $>$ and $<$ are defined. The comparisons $==$ and $!=$ are based on every single component of the tensor.

Use of the following functions should be avoided whenever possible since they use the expensive `eig(...)` functions to compute the eigenvalue, -vector decomposition and only return one value. It is probably better to create an eigensystem (instance of class `eigen.h`) and derive this measurements from there.

`float* eigvec1(float *vector) const;`	first eigenvector, corresponding to largest eigenvalue
`float* eigvec2(float *vector) const;`	second eigenvector
`float* eigvec3(float *vector) const;`	third eigenvector, corresponding to smallest eigenvalue
`float eigval1() const;`	first (largest) eigenvalue
`float eigval2() const;`	second eigenvalue
`float eigval3() const;`	third eigenvalue
`float anisotropy() const;`	anisotropy, see equation 4.17

A.2 eigen.h

The data stored in a eigensystem are three floating point precision values and three floating point precision arrays, which are all private members of `eigen.h`:
`float _lambda1, _lambda2, _lambda3;`
`float _vector1[3], _vector2[3], _vector3[3];`
The eigenvalues are sorted so that `_lambda1` is always the largest eigenvalue.

The constructors are:

`eigen(const float eig1, const float eig2, const float eig3, const float *vec1,`	
	a constructor which initializes all values
`eigen();`	a constructor which sets all values to zero.

To set values in an eigensystem there are several specialized functions:

`void set(const float eig1, const float eig2, const float eig3, const float *vec1,`	
`const float *vec2, const float *vec3);`	set all the values in an eigensystem.
`void set(const float eig1, const float eig2, const float eig3);`	
	set the eigenvalues.
`void setVectors(const float *vec1, const float *vec2, const float *vec3);`	
	set the eigenvectors.
`void setVector2(const float x, const float y, const float z);`	
	sets the second eigenvector to be $\hat{\mathbf{e}}_2 = (x, y, z)^T$.
`void resetVector3();`	sets the third eigenvector to $\hat{\mathbf{e}}_3 = \hat{\mathbf{e}}_1 \times \hat{\mathbf{e}}_2$
`void setZero();`	sets all the values to zero.

If the largest eigenvalue is zero then the whole eigensystem is zero. Therefore the last function only needs to set `_lambda1` to zero.

The single values can be accessed with:

`float value1() const;`	largest eigenvalue
`float value2() const;`	second eigenvalue
`float value3() const;`	smallest eigenvalue
`float vector1(const int index) const;`	returns element "index" (0|1|2) of eigenvector 1
`float vector2(const int index) const;`	returns element "index" (0|1|2) of eigenvector 2
`float vector3(const int index) const;`	returns element "index" (0|1|2) of eigenvector 3
`void print() const;`	sends the eigensystem to the standard output

The operators defined are:

`bool isZero() const;`	check if eigensystem is zero (true if so)
`bool operator==(const eigen &e) const;`	compares all elements of the eigensystem, no tolerance
`bool operator!=(const eigen &e) const;`	do.
`bool operator< (const eigen &e) const;`	less then, uses measurement specified in measure()
`bool operator> (const eigen &e) const;`	do.
`bool operator<=(const eigen &e) const;`	do.
`bool operator>=(const eigen &e) const;`	do.

Functions are:

`void scale(const float factor);`	scale eigensystem by "factor"
`void scale(const float x, const float y, const float z);`	scale differently in each direction
`void rotate(const float pitch, const float roll, const float yaw);`	Rotate eigensystem, the angles pitch, roll, and yaw have to be provided in degrees
`float det() const;`	determinant of the tensor $= \prod_{i=1}^{3} \lambda_i$
`float relation_Lambda1_Lambda2() const;`	absolute value of $\frac{\lambda_1}{\lambda_2}$
`float relation_Lambda1_Lambda3() const;`	absolute value of $\frac{\lambda_1}{\lambda_3}$
`void eigen2tensor(tensor &t) const;`	converts an eigensystem into a tensor
`int tensor2eigen(const tensor &t);`	converts a tensor in an eigensystem negative return value is an error, 1 is success
`float close_line() const;`	measurement of how close diffusion tensor shape is to a line (equation 4.13)
`float close_plane() const;`	a plane (equation 4.14)
`float close_sphere() const;`	a sphere (equation 4.15)
`float anisotropy() const;`	anisotropy measure describing deviation from the spherical case (equation 4.17)

Functions between two eigensystems:

`float cosinMaxEigen(const eigen &e) const;`	angle between the vectors of the maximum eigenvalues
`bool isSimilar(const eigen &e, const float tolerance, const int similarity) const;`	compare 2 eigensystem, return 1 if identical, 0 else
`bool hasSimilarShape(const eigen &e, const float tolerance) const`	do.
`bool hasSimilarSize(const eigen &e, const float tolerance) const;`	do.
`bool hasSimilarDirection(const eigen &e, const float tolerance) const;`	do.
`bool threshold(const float threshold=0.8) const;`	return true if eigensystem above threshold

A.3 basefield.h

Private members of the class `basefield.h` read the values either from the console or an xml-file:

`float readfloat(DOM_Node& node);`	read numerical values from node in xml file; return -1 if no value present
`int readstring(DOM_Node& node, char* mystring);`	read string from node in xml file; return -1 if no value present
	read attribute as string from node in xml file; return -1 if no value present pre:mystring is the name of the attribute to get value post: mystring is the value of the attribute;
`int readattribute(DOM_Node& node, char* mystring);`	
`void readfile(DOM_Node& node);`	read the location of the datafile from node in xml file (could be remo
`void readdata(DOM_Node& node);`	read the xml file and extract the data wanted;
`void readmanualy();`	read ALL values from console
`void readmanualy(int& value);`	read value from console, exit if entered -1
`void readmanualy(float& value);`	do.
`void readmanualy(char* value);`	do.
`void assert_values();`	check if all values to write xml-header are present if not, ask for values

Protected members that are only accessible to the derived classes are:

`int _dimx, _dimy, _dimz;`	the dimensions of the field
`float _deltax, _deltay, _deltaz;`	the distance between voxels = voxel dimensions
`char *_datafile;`	pointer to the data file
`char *_orientation;`	orientation of the field: axial, coronal, saggital
`char *_sourcefile;`	location of source to derive tensor file
`char *_originalfile;`	location of the original data
	baby specific data:
`int _gestationalAge;`	Gestational age of the baby in weeks
`int _mrNumber;`	Patient ID number
`int _studyNumber;`	Study number

```
int _seriesNumber;                          Series number for the diffusion data
char * _date;                               Date of the scan
```

Any new element added to the DTD should be added here.

Public members, i.e. functions accessible by any program are:

```
                                            do everything needed to call readdata, should not need
void read_XMLHeader( const char xmlFile[]);
                                            any editing when changing DTD
void write_XMLHeader(const char xmlFile[]);
                                            write a xml-header file at location xmlFile

                                            set values
void setNumbers(const int age, const int patient_id, const int studyNr,
void setDate(const char* date);
void setOrig(const char* orig);
void setOrientation(const char orientation);

                                            set the voxel dimensions for field
void setDelta(const float deltax, const float deltay, const float deltaz);
                                            get the voxel dimensions for field
void getDelta(float& deltax, float& deltay, float& deltaz);
                                            get the volume dimensions for field
void getDimension(int& dimx, int& dimy, int& dimz);

char * getPath(char * path);                get the path to the data without any extension
```

The actual path to the data is generated in the derived class by appending the corresponding extension.

No function to set the volume dimensions is provided, since this is either already defined by the data set and therefore in the xml-header or a new data set is created and then the corresponding derived class has to set this values.

Finally three generic functions are included in this class:

```
                                            check boundaries for offset and window-size
                                            and reset them if necessary.
                                            used for display purposes:
                                            Plane rotates the window depending on window-size
                                            plane[3] = 2: Image is in xy-plane
                                            = 1: Image is in xz-plane
                                            = 0: Image is in yz-plane
                                            in each case plane[0] and plane[1]
                                            are the xy dimensions in the image coordinate system
                                            plane[2] the number of images
void check_boundaries(int offset[3], int windowsize[3], int plane[4]=NULL) const;

short * read_pic(short *image,char path_file[], int offset=0);
unsigned short * read_pic(unsigned short *image, char path_file[], int offset=0);
```

The last two functions read the content of the *.pic files generated by `LSDI_recon`.

A.4 convert

The program `convert` is used by typing:
`./convert -i=infile -o=outfile [options]`
where infile and outfile are the full path and filename to the data. No file extensions are needed and the conventions described in section 5.2 are used: tensor data has the extension .ten, eigen data .eig and the background .bg. In case the type of the infile is "raw" then the infile is the path and the name of the files from which to derive the tensors, but without the ending XXX.pic. As output the data as well as an xml-Header is written, as long as an eigensystem is involved in the output. Table A.1 explains the options for `convert`.

Call	Default	Description
-all	-it=raw	convert from raw to tensor AND eigen
-it=(raw|tensor|eigen)	raw	type of infile
-ot=(tensor—eigen)	tensor	type of outfile
-ix=	256	x-dimension of infile
-iy=	256	y-dimension of infile
-iz=	1	z-dimension of infile
-ox=	same as input	x-dimension of outfile
-ox=	same as input	y-dimension of outfile
-oz=	same as input	z-dimension of outfile
-dx=	1	x-voxel dimension
-dy=	1	y-voxel dimension
-dz=	1	z-voxel dimension
-offx=	0	x-offset for outfile
-offy=	0	y-offset for outfile
-offz=	0	z-offset for outfile
-id=	0	Patient ID
-ex=	0	Examen number
-ser=	0	Series number
-ag=	0	Age
-or=(s|c|a)	a	Orientation: saggital, coronal or axial
-orig=	notAvailable	location of the original data
-date=		actual date (default is empty)
-r=	-1	remove strips, 0 outside the object 1 outside and inside the object

Table A.1: Parameters for the program `convert`

A.5 rigidreg

`rigidreg` is simple to use. It only takes three arguments:
`./rigidreg moving-xml-file stationary-xml-file interpolation-method=(1|2)`
the interpolation method can be either linear (1) or nearest neighbor (2) as described in Section 6.2.

A.6 nonrigidreg

The program `nonrigidreg` is used by typing:
`./nonrigidreg imagetype [options] [processing-options]`

where imagetype is a value vetween 0 and 4, which selects the type of input and output and is described in table A.2. All the options with their default values are shown in table A.3

0	synthetic image, synthetic displacement; no input needed
1	MRI input image, synthetic displacement
2	MRI input and output image
3	xml input representing tensordata, synthetic displacement
4	xml input and output representing tensordata

Table A.2: Image type for nonrigid registration

Call	Default	Type	Description
-i=	movingfile	char array	MRI or xml-file
-s=	stationaryfile	char array	MRI or xml-file
-o=	outputfile	char array	path where to save output, will be same of same type as input
-syn=	none	char array	path to synthethic displacement fields saved as floats
-x=	256	integer	x-dimension for MRI file
-y=	256	integer	y-dimension for MRI file
-z=	1	integer	z-dimension for MRI file
-mw=	7	inpair integer	size of moving window=window measuring correlation
-sw=	7	inpair integer	size of stationary window=window where match is searched
-m=	0	0 or 1	$\left\{ \begin{array}{l} 1 = \text{least square error (equation 7.18)} \\ 0 = \text{max. crosscorrelation (equation -7.17)} \end{array} \right\}$ matching method
-mm=	0	0, 1, 2	$\left\{ \begin{array}{l} 0 = \text{none} \\ 1 = \text{linear} \\ 2 = \text{gauss} \end{array} \right\}$ weighting of matching window
-sc=	1	integer	number of scales to match
-scf=	2	integer	scale factor for multiscale matching (only if sc>1)
-sl=	0	integer	gauss-filterlength before downsampling
-n=	3	inpair integer	neighborhood where expectance of derivatives is computed
-st=	0.01	float	threshold for σ in the point-detection
-pt=	0.0005	float	threshold for extracted points
-pm=	0	0, 1, 2, 3, 4	$\left\{ \begin{array}{l} 0 = \frac{c \cdot det()}{(trace + \sigma \cdot max(trace()))} \text{ (equation 7.13)} \\ 1 = \frac{det()}{trace()} \text{ (equation 7.12)} \\ 2 = trace() \text{ (equation 7.14)} \\ 3 = \text{2+local maximum in a 5 neigborhood} \\ 4 = \text{2+local maximum in a 9 neighborhood} \end{array} \right\}$ point extracting method
-k=	9	integer	number of neighbors for kriging interpolation
-km=	0	0, 1, 2, 3, 4	$\left\{ \begin{array}{l} 0 = \text{Linear} \\ 1 = \text{Spherical} \\ 2 = \text{Exponential} \\ 3 = \text{Cubic} \\ 4 = \text{Gaussian} \end{array} \right\}$ method for variogram in kriging
-a=	0	0, 1	$\left\{ \begin{array}{l} 0 = \text{no} \\ 1 = \text{yes} \end{array} \right\}$ anisotropic diffusion before pointextraction
-astep=	2	integer	time step for anisotropic diffusion
-aiter=	10	integer	number of iterations in anisotropic diffusion
-acon=	0.5	float	contrast value for anisotropic diffusion
-anoise=	0.8	float	noise value for anisotropic diffusion
-g=	1	3, 5, 7	Gauss-filterlength (after anisotropic diffusion, before pointextraction)
-d=	10	integer	for maximum displacement in synthetic random displacement
-r=	10	integer	>=d ratio of pixels in sparse synthetic random displacement
-grid=	32	integer	size of synthetic grid
-border=	64	integer	size of border in synthetic grid
-ran=	123456	integer	to initialize random numbers
-l=	1	integer	loop for applying matching several times
-tw=	1	0, 1, 2	$\left\{ \begin{array}{l} 0 = \text{only rotation (equation 7.30)} \\ 1 = \text{warping without scaling (equation 7.29)} \\ 2 = \text{full warping (equation 7.28)} \end{array} \right\}$ local tensor warping

Table A.3: Parameters for the nonrigid registration program

A.7 Document Type Definition

```
<!ELEMENT spl_tensor (header, data)>
<!ATTLIST spl_tensor version CDATA #REQUIRED> <!-- the version of this DTD -->

<!-- Header Section -->

<!ELEMENT header (mrNumber?, studyNumber?, seriesNumber?, age?, original?, source,
file)>

<!ELEMENT mrNumber (#PCDATA)> <!-- MR number -->
<!ELEMENT studyNumber (#PCDATA)> <!-- study number -->
<!ELEMENT seriesNumber (#PCDATA)> <!-- series number -->
<!ELEMENT age (#PCDATA)>
<!ATTLIST age type (normal | gestational) "normal"> <!-- babies:  gestational age -->

<!ELEMENT original (#PCDATA)> <!-- path where the data was graped from -->
<!ATTLIST original date CDATA #IMPLIED> <!-- when the data was graped -->
<!ATTLIST original datatype (short | float | double) "short">
<!ATTLIST original byteorder (LSB | MSB) "MSB" >
<!ATTLIST original orientation (axial | coronal | saggital) "axial">

<!ELEMENT source (#PCDATA)> <!-- path where the data was loaded -->
<!ATTLIST source date CDATA #IMPLIED> <!-- when the data was loaded -->
<!ATTLIST source datatype (short | float | double) "short">
<!ATTLIST source byteorder (LSB | MSB) "MSB" >
<!ATTLIST source orientation (axial | coronal | saggital) "axial">

<!ELEMENT file (#PCDATA)> <!-- path where the actual file resides, no extensions -->
<!ATTLIST file date CDATA #IMPLIED> <!-- when the file was first created -->
<!ATTLIST file datatype (short | float | double) "float">
<!ATTLIST file byteorder (LSB | MSB) "MSB" >
<!ATTLIST file orientation (axial | coronal | saggital) "axial">

<!-- Data Section -->

<!ELEMENT data (dimensions, description?)>

<!ELEMENT description (#PCDATA)>

<!ELEMENT dimensions (volumesize, voxelsize)>
<!ELEMENT volumesize (dim_x, dim_y, dim_z)>
<!ELEMENT voxelsize (delta_x, delta_y, delta_z)>

<!ELEMENT dim_x (#PCDATA)>
<!ELEMENT dim_y (#PCDATA)>
<!ELEMENT dim_z (#PCDATA)>
<!ELEMENT delta_x (#PCDATA)>
<!ELEMENT delta_y (#PCDATA)>
<!ELEMENT delta_z (#PCDATA)>
```

Example xml-Header

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE spl_tensor SYSTEM "/home/rsierra/c++/tensor/data/spl_tensor.dtd">

<spl_tensor version="1.0">
   <header>
      <original date="11.2000" orientation="axial">/d/stockholm/data/diffusion
                /highresdata/axial</original>
      <source date="11.2000" orientation="axial">/projects/shortterm/warfield
                /rsierra/examples/multislice/westin/axial/tensor/axial-S</source>
      <file date="29.12.2000" orientation="axial">/projects/shortterm/warfield
                /rsierra/examples/mydata/westin</file>
   </header>
   <data>
      <dimensions>
         <volumesize>
            <dim_x>256</dim_x>
            <dim_y>256</dim_y>
            <dim_z>24</dim_z>
         </volumesize>
         <voxelsize>
            <delta_x>0.859375</delta_x>
            <delta_y>0.859375</delta_y>
            <delta_z>2.500000</delta_z>
         </voxelsize>
      </dimensions>
   </data>
</spl_tensor>
```

Appendix B

Example Images

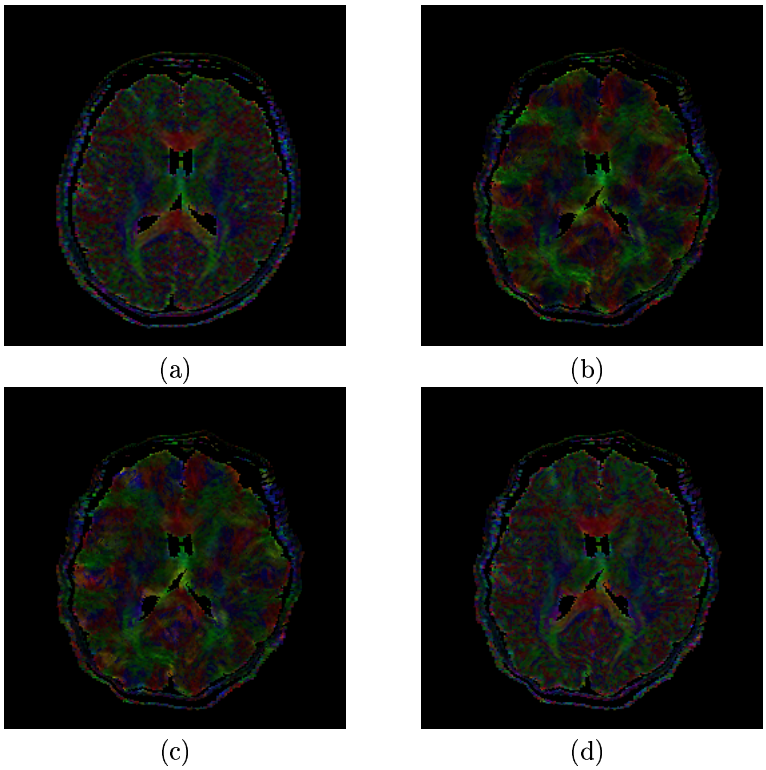

Figure B.1: Local transformations of tensors after synthetic displacement. (a) Original image; (b) Full local transformation, Equation 7.28; (c) Local transformation without scaling, Equation 7.29; (d) Only local rotation, Equation 7.30.

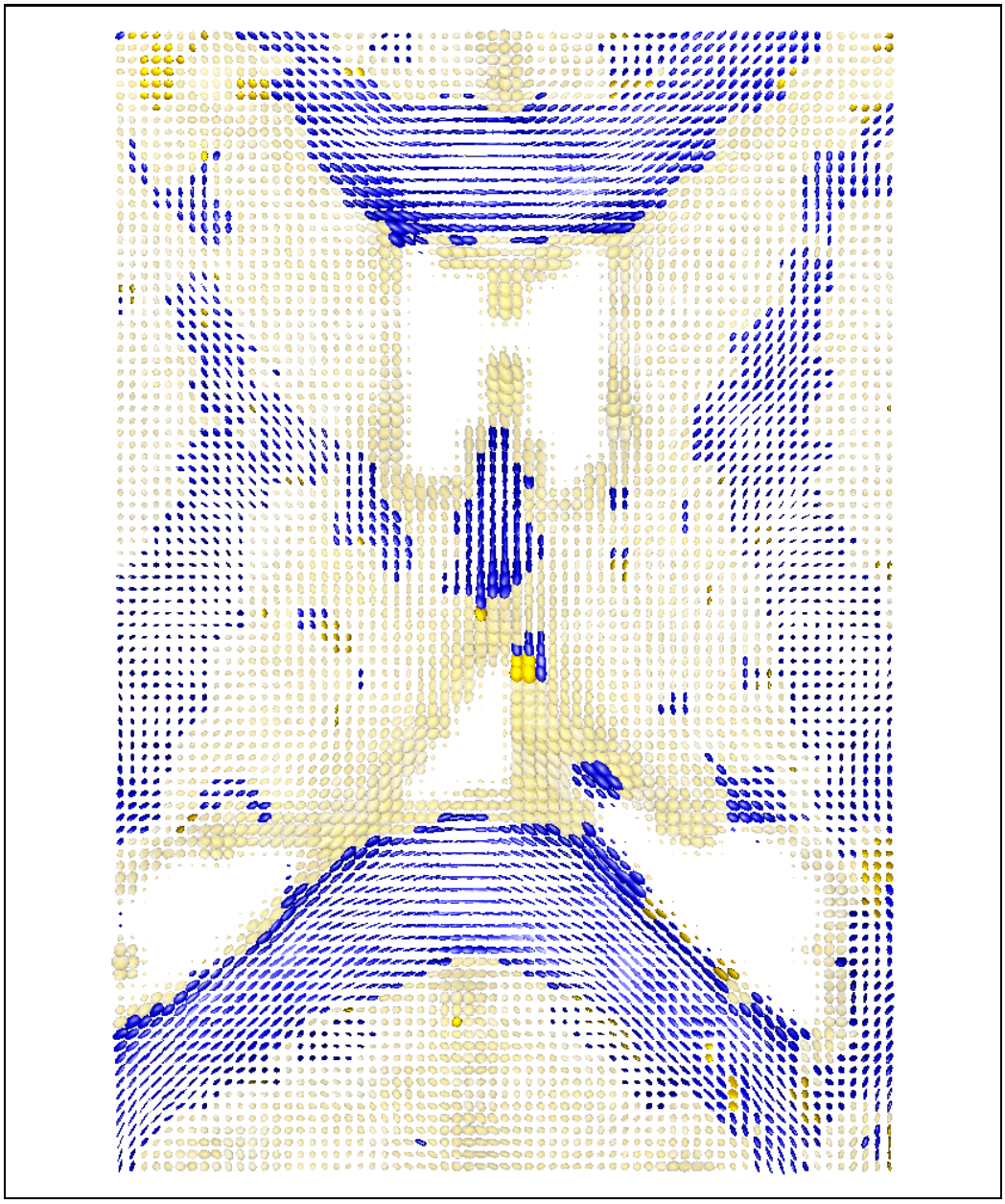

Figure B.2: Diffusion tensor image of an adult human brain. The tensors are color encoded as described in section 4.3 on page 15. The image shows a field of 80 × 40 × 1 tensors.

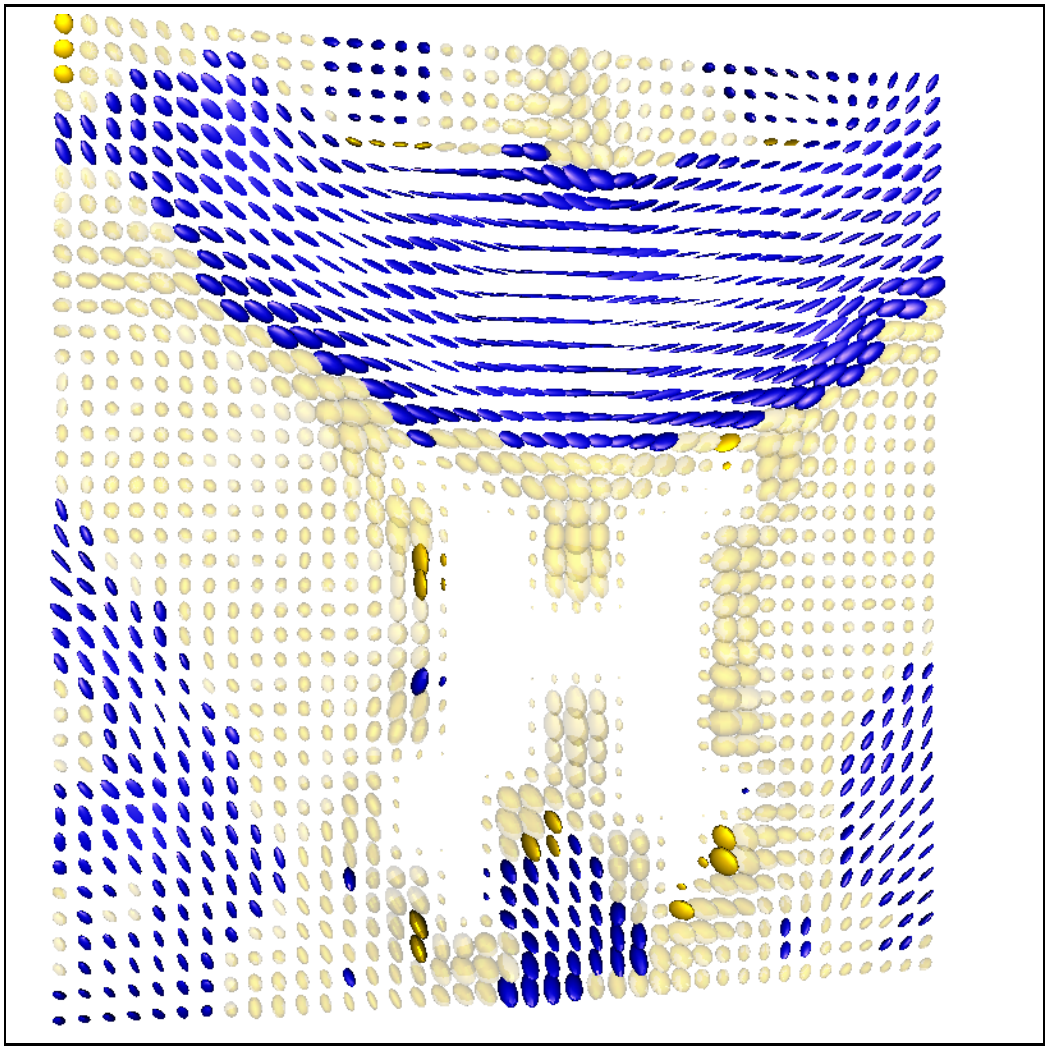

Figure B.3: Diffusion tensor image of the corpus callosum of an adult human brain. The tensors are color encoded as described in section 4.3 on page 15. The image shows a field of $40 \times 40 \times 1$ tensors.

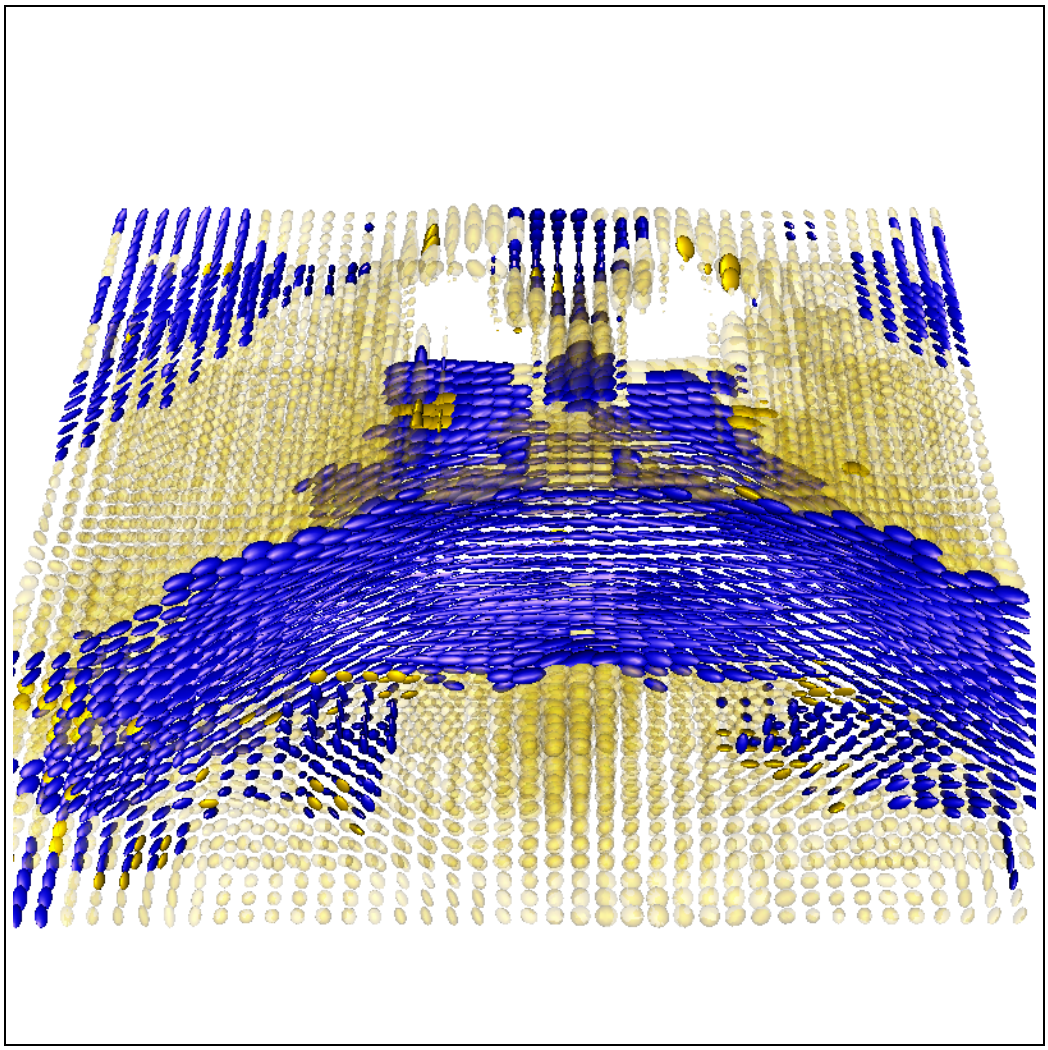

Figure B.4: Three dimensional representation of the frontal lobe of the corpus callosum in an adult human brain. Field of view is 40 × 40 × 4, smoothed with a Gauss filter of length 3

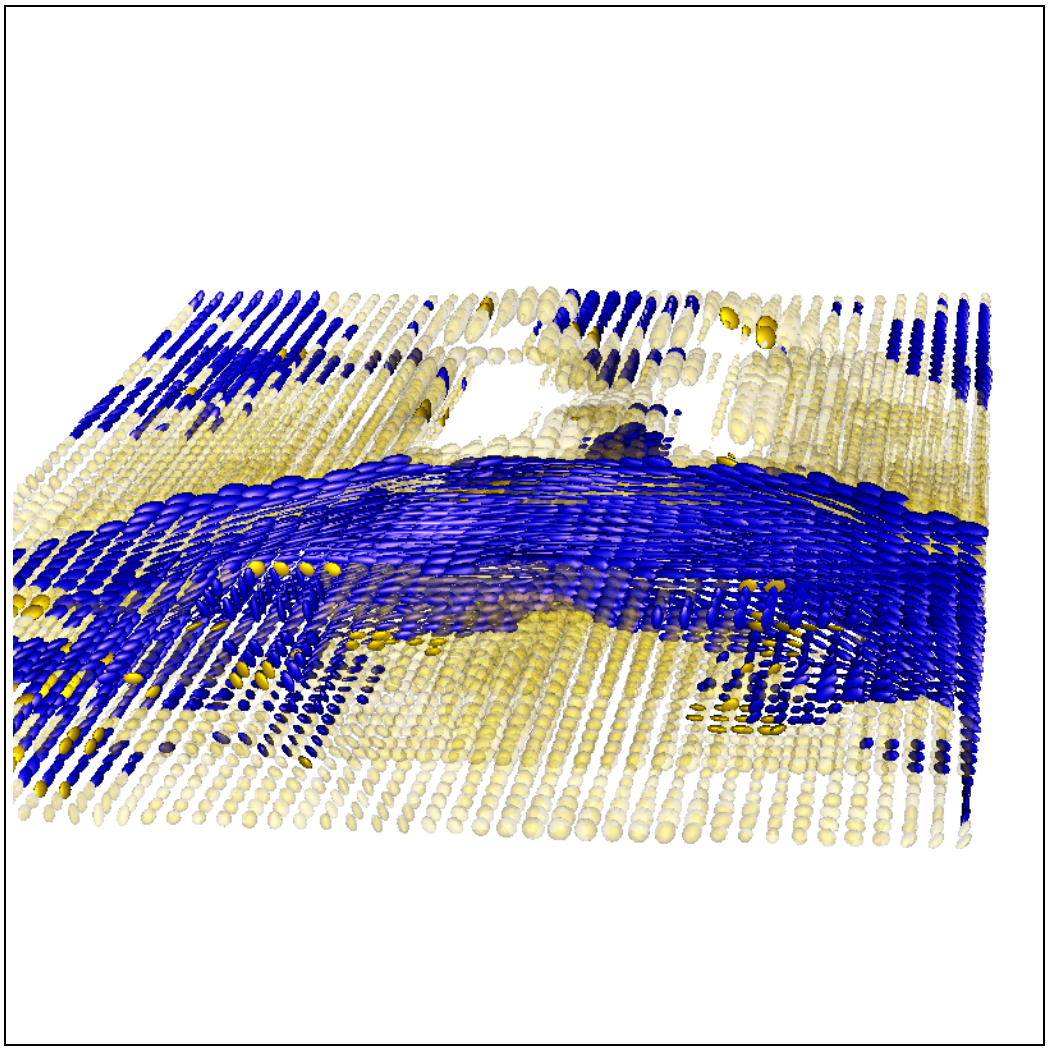

Figure B.5: Three dimensional representation of the frontal lobe of the corpus callosum in an adult human brain. Field of view is 40 × 40 × 4, smoothed with a Gauss filter of length 3