FIONA

A tailored fMRI image processing pipeline

Yidi CAO

Supervised by:
Emile Hendriks, from TU Delft
S.F.W. Neggers, from Brain Science Tools BV

TU Delft

FIONA

A tailored fMRI image processing pipeline

by

Yidi CAO

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday August 30, 2016 at 14:00 PM.

Student number: 4527313
Project duration: March 7, 2016 – August 30, 2016
Thesis committee:

Prof. Dr. Emile. Hendricks,	Pattern Recognition & Bioinformatics, TU Delft
Dr. S.F.W. Neggers,	Brain Center Rudolf Magnus, UMC Utrecht
Dr. Anna Vilanova,	Computer Graphics & Visualization, TU Delft

This thesis is confidential and cannot be made public until December 31, 2016.

TU Delft

Preface

This project was done under help from many people. Thanks Dr. Emile Hendriks for patient supervision. Thanks Dr. Bas Neggers, Ms. Petar Petrov and Dr. Daan Baas for daily instructing, supervising and inspiring. Thanks Myriam Coes for suggesting, sharing and collaborating. Thanks Dr. Anna Vilanova for being a committee member. Thanks Erik Jansen for coordinating. Thanks my family for endless supporting.

Yidi CAO
Utrecht, August 2016

Abstract

In neuroscience research, fMRI image analysis is a major approach to identify the relationships between brain region and behavior. fMRI provides delayed real time brain activity signals non-invasively. Spatial preprocessing and statistical analysis needs to be performed to find the brain region that is responsive to certain tasks or stimulations. This project introduce an implementation of a tailored fMRI image processing pipeline, FIONA (Functional Imaging Overlay for NAvigated TMS), based on Insight Toolkit (ITK). The major functionalities of the software include preprocessing (realignment, co-registration, spatial smoothing), GLM analysis, hypothesis testing and multiple comparison correction (mainly FDR). FIONA is designed for usual clinical users or fMRI analyst who may not be an expert in neuroscience. It is designed for navigated TMS as its intended use. We also did benchmarking on different ITK optimizers comparing their performance in realignment section, to select an overall best optimizer. We test our whole implementation on different level noisy fMRI datasets.

Contents

1 Introduction

fMRI images are widely used in neuroscience research. However, fMRI images directly from an MRI scanner can not be interpreted directly. fMRI processing is needed to see the connection between brain activation regions and behaviors. Our project FIONA, is a fMRI processing software based on the Insight Toolkit (ITK), and intended to be used in stereotactic neural navigation of TMS (see section 1.3). In this chapter, we will introduce some basic concept and background around FIONA.

1.1. Functional MRI

fMRI for functional magnetic resonance imaging, is a neuroimaing method, that detects brain activities non-invasively. Traditional MRI registers anatomic structure, by detecting the density of water molecules. However, fMRI observes changes of blood flow in a brain, to detect brain activity. This measure of oxygen level in blood flow is called *BOLD*, which stands for *Blood-Oxygen-Level Dependent*. fMRI records the BOLD changes in real-time, or as in real-time as possible. There is a trade-off, compared to traditional MRI images, between spatial and temporal resolution. A typical structural scan of MRI takes about 8-10 minutes and may have a spatial resolution of less than $1mm$. fMRI compensates spatial resolution for temporal resolution, and hence the spatial resolution of fMRI is comparatively low, around $3mm$.

Nowadays fMRI is widely used in many aspects of neuroscience research. One major application is to determine which area on the cortex is related to a behavior. Previous research showed that the area in the brain is more activated when its BOLD signal is higher. It is intuitive that this area need more oxygen to support its metabolism. By analyze BOLD signals along time, together with task designed in the experiment, we can see the correspondence in time.

1.2. FIONA

Our project, FIONA, short for *Funtional Imaging Overlay for NAvigated TMS*, is a fMRI processing application under development at the company Brain Science Tools BV. It is designed to have two basic functional modules: *Pre-Processing* and *GLM Analysis*.

The main intention of developing FIONA, is to develop a simplified fMRI processing software, that is easy to use for practical clinical users and fMRI analysts, who often do not have sufficient background in neuroscience and image processing. FIONA allows clinical users to execute the whole pipeline of fMRI processing and analysis in a few steps, where the pipeline makes the most difficult decisions for the user. That is, if the user provides fMRI images and minimal and easy to understand information on the experimental paradigm used for the fMRI session. FIONA is able to generate a statistical parametric map (SPM), showing brain-behavior relationships, for a summary on the GLM approach to generate an SPM map of fMRI activation.[10]

Also, FIONA as a project at Brain Science Tools BV is designed to be used in conjunction with the Neural Navigator. The Neural Navigator is the main product from Brain Science Tools BV that is used to stereotacti-

cally navigate the coil in Transcranial Magnetic Stimulation sessions (See 1.3 below for more an introduction into TMS), guided by MRI and fMRI scans. We designed and specified FIONA to meet the requirement of the Neural Navigator.

1.3. TMS & Neural Navigator

The project is part of my internship at *Brain Science Tools BV*, Utrecht, The Netherlands. Brain Science Tools BV is an innovative company specializing in MRI related technology and neuro-stimulation equipment for neuroscience labs and clinical users. They also offer advice, training and courses in brain imaging and TMS.

TMS is short for Transcranial Magnetic Stimulation, which is a technique stimulating a small area on the cortex, by a strong but short transcranial magnetic impulse generated from a coil [2]. TMS now is used in some aspects of diagnosis or treatment. The major use of TMS is to treat depression [9], which is the only use of TMS that is approved by FDA. Moreover, as introduced by [12], TMS is now starting to be used to improve the recovery of function after stroke. [4] shows rTMS is effective in neuropathic pain treatment. Also, TMS is used in clinical research to test the connection between some area on primary motor cortex and a corresponding muscle, especially in cases of stroke, multiple sclerosis, amyotrophic lateral sclerosis, movement disorders and so on.

TMS requires coil placement over a brain region with millimeter accuracy. Neural navigation is the technical solution allowing MRI guided TMS coil placement. Neuro-navigation displays the TMS coil and the patients head and brain in real time on a screen, guiding the TMS coil to the planned location in the brain for optimal placement. Neuro-navigation can increase the accuracy of target location and the repeatability of the stimulation foci and motor evoked potential amplitudes compared to non-navigated TMS coil placement. This is particularly important for rTMS protocols with daily sessions for several weeks. For example, in the treatment of depressions, clinicians wish to target the Dorsolateral Prefrontal Cortex (DLPFC) in order to alleviate symptoms [9]. Treatment effects are much larger when DLPFC is targeted based on an individual MRI scan [17]. Without MRI guidance, DLPFC is often missed by several centimetres rendering the treatment ineffective.

The neural navigator software (NeNa: http://www.neuralnavigator.com/) of Brain Science Tools supports built-in pre-processing of structural MRI data, such as image registration and segmentation. Functional MRI activation maps from packages such as SPM, FSL and AFNI can be loaded (for investigational purposes), but for the clinical workflow an image processing algorithm for fMRI image registration and time analysis still need to be build and tested. Because users with a clinical background frequently indicated they desire a workflow, also for fMRI processing, that is native to the application or easy to use in conjunction with it, as they are not comfortable using research grade software. Also, packages as SPM, FSL and AFNI cannot legally be used for clinical purposes in most countries. Therefore, Brain Science Tools aims to develop and certify medical grade fMRI processing software itself, in compliance with the IEC62304 industry standard for medical device software. FIONA is a way to explore the feasibility of this ambition, and in general the steps required in IEC62304 standard are followed.

Our project FIONA is designed to optimise the image analysis pipeline, build a graphic user interface and test and document the developed software according to the aforementioned IEC62304 industry standard.

1.4. SPM

Statistical Parametric Mapping (*SPM*) is originally a MatLab toolbox for statistical processes used to test hypotheses on fMRI or other brain imaging data, developed by *Wellcome Trust Centre for Neuroimaging, UCL, London* and released under the General Public License (GPL) for use in academic research. The latest release can process not only fMRI images, but also PET, SPECT, EEG and MEG.

For our implementation SPM is used as a reference to test performance of our software against, in realignment, co-registration, GLM analysis and FDR thresholding. FIONA is certainly not a copy of SPM nor does it contain any source code derived from SPM. FIONA is designed to be used in conjunction with the *Neu-*

ral Navigator (NeNa), therefore, we implement those functionalities that we actually need for fMRI guided coil placement in practice. For details on the use of (f)MRI for TMS coil placement see several papers by Neggers et al [15]. For example, we did not apply normalization in FIONA, which a common procedure in a typical fMRI preprocessing, because NeNa does not need the brain to be registered to a standard brain. More importantly, typical normalization involves nonlinear deformations that change the shape of the head [7], which renders stereotactic use of the resulting images impossible, as stereotactic navigation should only deploy rigid body registrations between a patients head in the TMS setup and his/her MRI scan. SPM and other MRI processing pipelines typically divide the whole process into different procedures. Users need to be very well-informed about what they need to do in each step and need to be familiar with most scientific and mathematical details of these procedures. Also, users need to find files generated from one function and import them into another. FIONA users will only need to import fMRI images from scanner and enter experimental design informations to generate a parametric map. Most detailed choices are embedded in the software and automized, and are guided by typical use cases in fMRI guided TMS.

1.5. Overview

The paper is structured in the order of typical fMRI image processing pipeline. In chapter 2, we introduce the basic design of our software, FIONA. We will introduce the functionality requirements and system structure of the design, and also the development tools we used in our implementation. The rest part of the paper is structured around subsequent parts of typical fMRI processing. At first, in chapter 3, we introduce fMRI preprocessing part consisting of 3 main parts: Realignment, Co-registration and Smoothing. For realignment, we adopted different optimizers with different strength and weaknesses and compared their performance in a benchmarking approach. The experiments and results of this benchmark are discussed in chapter 4. The statistical analysis part consists of a basic General Linear Model method and some basic statistical thresholding taking into account the multiple comparison problem, especially false discovery rate (FDR), which is described in Chapter 5 and 6. A conclusion of the entire project is presented in chapter 7.

2

The Software Design

The FIONA software is written in C++, using the Microsoft Visual Studio compiler and associated libraries and depends on the Insight Segmentation and Registration Toolkit (ITK: https://itk.org/) for image processing and on the Visualization Toolkit (VTK: https://vtk.org/) as a visualization library and the QT set as the user interface library. The FIONA runs on normal PCs with a 32 or 64 bit infrastructure. In this chapter we introduce some brief information about our software FIONA, including the software design and the SRS.

2.1. Software Design

The main functions of the FIONA as listed above are each divided into 3 logical major software units or modules, governing the design of the entire software project. All classes defined in the project are used in either one of these main units (we refer to them as 'Managers'). Below a list and a graphical representation of the relationships between the units are outlined.

Pre-Processing This unit defines and implements the pre-processing part of fMRI analysis pipeline. It performs realignment (re-slicing optional), co-registration and spatial smoothing. It takes MR images from scanner as input and generate new NIfTI images as output. This unit depends on *ITK* library.

GLM This unit defines and implements the General Linear Model computation and thresholding methods in fMRI statistical analysis. This unit follows Registration unit. It takes realigned images as input, along with user defined model, computes b-maps and t-map based on user's hypothesis and reports significance. This unit depends on *ITK* library.

Window Manager This unit defines and implements the user interface, consisting of the main window, menu and tabbed interfaces. It processes all callback messages generated by the user and then calls proper slot functions. This unit also implements visualization methods. This unit depends on the *Qt* and *VTK* libraries.

2.2. Functional Requirements

In software industry, a so-called *Software Requirement Specifications* (SRS) is needed for a software product. FIONA is medical device software that processes fMRI data resulting in a 3D rendering of the activation patterns and processed images. This process consists of pre-processing (realignment, co-registration, spatial smoothing), fitting the GLM and creating an activation map. According to the demand, we defined the main and optional functionalities of FIONA, listed here:

- **Loading of (f)MRI Data**
  The FIONA software shall be able to load MRI data stored in the Nifti 1.0 and DICOM format into RAM.

- **3D Surface Rendering of MRI Data**
  The Software package shall surface render MRI data, using different presents for anatomical scan, segmented gray matter scan, and fMRI activation map, and display it in 3D in the main render window. Triangulation of MRI tissue intensities is to be used. The segmented image could be set color-coded by statistical maps.

- **Change Nifti Header**
  The FIONA software shall be able to change information in a NifTi header. NifTi header information should be loaded and changes and then stored. The changed parts in NifTi header are the sform_code, srow_x[4], srow_y[4] and srow_z[4].

- **Save Nifti file**
  The FIONA software shall be able to save images as a NifTi file including the NifTi header and 4 bytes extender.

- **Realignment**
  The FIONA software shall be able to perform re-alignment of fMRI data. The realignment is performed between images in the same fMRI series. Following images are registered to the first image. For each image the translation (in x, y, z direction) and rotation (roll, pitch and yaw) with respect to the first image is determined. Realignment apply the transform on the series of images to make the brains being in a same spatial position. These six parameters are stored in a *.txt* file.

- **Re-Slice**
  The FIONA software shall be able to re-slice fMRI images. During the procedure of re-slice, a series of registered images will be matched voxel-to-voxel to the first image resulting in a new series of images.

- **Co-Registration**
  The FIONA software shall be able to coregister functional fMRI images to structural MRI images. Co-registration is performed in two stages. First a translation is performed and then a 3D Euler transform. The translation (in x, y, z direction) and rotation (roll, pitch and yaw) of the anatomical image in respect to the first image fMRI image is determined.

- **Spatial Smoothing**
  The FIONA software shall be able to perform spacial smoothing on fMRI images. Spacial smoothing is performed using a discrete Gaussian kernel. Spacial moothing can only performed in combination with reslicing.

- **Importing Models**
  The FIONA software shall be able to import different models for use in GLM. The software is able to interpret the text file and convert the file into usable input for the GLM algorithm.

- **Generate Models**
  The FIONA software shall be able to generate simple models for use in GLM. By asking for the onset and TR, a GLM model is created.

- **Import Contrast Vector**
  The FIONA software shall be able to import contrast vector for hypothesis testing in GLM. The software is able to interpret the text file and convert the file into usable input for the GLM algorithm. Also, to ordinary clinical users, FIONA will ask some simple question and auto-generate a contrast vector for them.

- **Generate Contrast Vector**
  The FIONA software shall be able to generate a contrast vector for hypothesis testing in GLM. FIONA shall have an intuitive tool for making contrast vectors.

- **Calculate β-Maps**
  The FIONA software shall be able calculate β-value maps for each model imported. The β-value maps are save in NifTi files as well.

- **Calculate t-Maps**
  The FIONA software shall be able calculate t-value map with provided contrast vector or proposed hypothesis, and degrees of freedom. The t-value map is save in NifTi files as well.

- **Compute Threshold**
  The FIONA software shall be able compute a threshold for T-map, to see the significance. The threshold can be set by 3 different methods: setting plain threshold value; computed from Bonferroni correction; and false discovery rate method.

- **Apply Threshold**
  The FIONA software shall be able to apply the computed threshold on the t-map.

- **Visualize**
  The FIONA software shall be able visualize the t-map with realigned or segmented anatomical brain structure (from T1 image), with computed threshold.

2.3. Regulatory Requirements

The regulatory and statutory requirements applicable to medical device software need to be verified upon software testing by verifying completion of current software documents in the design history file. This ensures that the regulatory requirements according to the medical device directive (MDD 93/42/EEC) that applicable to standalone medical devices, NEN-IEC 62304:2006 (Medical device software – Software life-cycle processes), are met during release of the software.

3

Pre-Processing

In fMRI research, a most common target is to identify which area on cortex is activated because of a related task. However, the images directly from the scanner has too much noise and defects. A pre-process should be done before the dataset is going to statistical analysis. A typical pre-processing of fMRI data contains 6 steps: *Slice-Timing Correction, Realignment, Co-Rgistration, Normalization, Spatial Smoothing, Temporal Filtering*[1].

In our project, some of these steps are applied and some of them not, according to special demands. We will discuss them in detail in coming sections. A typical pipeline of fMRI pre-processing is shown in 3.1. Green blocks are modules that we implemented in our project, yellow ones are that we did not apply.

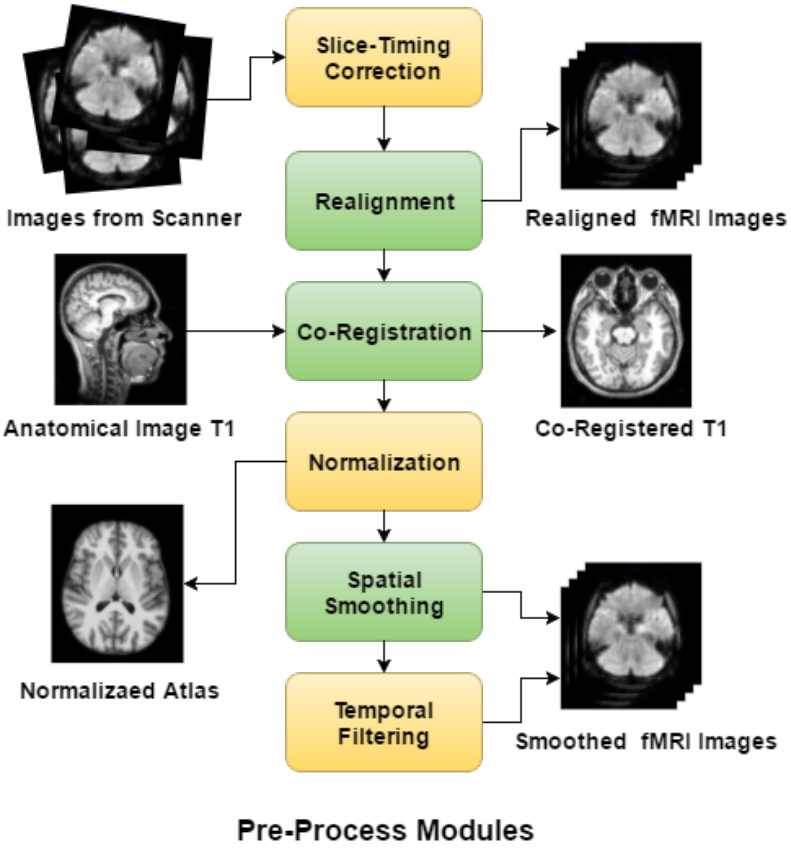

Figure 3.1: Pipeline of fMRI Images Pre-Processing in Our Project

Apart from the 6 steps above, there are other optional steps that are less commonly used, for example, *Quality Assurance*, *Distortion Correction*, *Grand Mean Scaling*. They are not implemented in our project and we are not going to discuss them in this paper.

3.1. Slice-Timing Correction

MRI data are collected in slices. Every slice is a 2-D array of pixels. A typical spatial resolution of fMRI in z-axis is like $3.25mm$. Therefore, a scan of a whole brain may takes about 33 slices in total. These 33 slices are actually not scanned at a same time, although they are combined into one 3-D image. The time between 2 scans is called *Temporal Resolution* or *TR*. A typical TR is about *2.5s*. There is a trade-off between temporal and spatial resolutions. Compared to traditional MRI, fMRI sacrifice spatial resolution for a better temporal resolution.

Slice-Timing Correction is a step to compensate the differences in slices of a 3-D volume. Usually we use a interpolation to compute the intensity of every slice, at the time when first slice is scanned. The interpolator can be various, from linear to *sinc*. See 3.2.

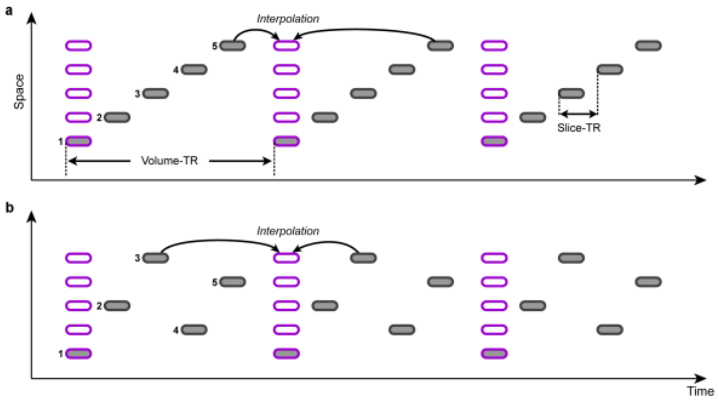

Figure 3.2: Slice-Timing Correction by Interpolation, Image from *Brain Voyager*

However, slice-timing correction is less used in recent fMRI researches. We did not apply slice-timing correction as well. The main reasons lies in three. First is that the scanning speed of MRI machines is much higher today than it was. The effect from differences in slices decreases. Second, the interpolation computation may induce much noise, that makes the harm of the correction more than it helps, especially when a motion occurred during a TR. Third, in our testing datasets, all experiments are of block designs. In block-design experiments, examinees are asked to do a task for several minutes, which make the correction unnecessary. However, in event-related designs, the slices differences still matters.

3.2. Realignment

Realignment or *Registration* of fMRI images is the first core functionality in our project. Realignment is a task to match two images into one coordinates system, by a spatial transform. In coming statistical analysis, all computation are done voxel-wisely, which demands a series images to share a same coordinate grid. Realignment is the most important and indispensable step in fMRI pre-processing. According to Huettel et al. (2004), very slight movement may ruin a dataset badly. He gave an example that a movement of head in $5mm$, influence the activation values by a factor of 5. Normally the result of registration should make the movement on cortex less than $1mm$, as the spatial resolution of fMRI images is like $3mm$.

Usually consecutive images are registered to the first image or the mean image. In our implementation, we realign following images to the first. We developed our software in Insight Toolkit (ITK), which provided fantastic C++ libraries for image segmentation and registration, especially for medical images. In ITK, the registration process are divided into 6 major blocks:[6]

- **Target**, which we usually name as *Fixed Image*, is the image that we are registered to. In our implementation, we select first image in time series as Target.
- **Reference**, which we usually name as *Moving Image*, is consecutive images in a time series. Their spatial information is varied and we need to match them to our target by applying a spatial transform.
- **Transform**, is the output of registration. A spatial transform alters Moving Image to Fixed Image.
- **Metrics**, is similarity measures to evaluate how well is transformed Moving Image matches Fixed Image. In ITK, a metric object can be set as a cost function to an optimizer.
- **Interpolator**, is how we compute intensities at non-grid point.
- **Optimizer**, is the mathematical algorithm to update parameters in a transform iteratively, to minimize a cost function. In image registration cases, the cost function could be metric objects.

In ITK architecture, there are 4 representations of data: *Image*, *Point Set*, *Mesh* and *Spatial Object*. In our implementation, we represent our fMRI images in itk::Image. ITK provided many registration approaches and we adopted itk::ImageRegistrationMethodv4 for our intensity-based realignment. In ITKv4 registration framework, computations are all happen in a physical grid rather than a fixed image grid. That is, both fixed image and moving images are re-sampled in a so-called *virtual image*. Metrics and their optimization are done in the virtual image domain. The framework of ITKv4 registration is shown in 3.3

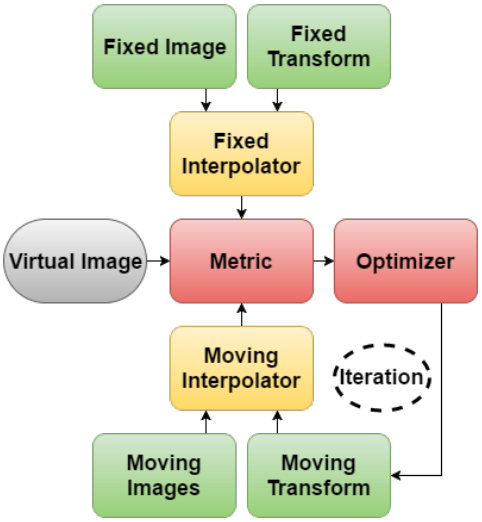

Figure 3.3: ITKv4 Registration Framework

In following subsections, we will discuss different transform prototypes, similarity metrics, optimization algorithms and interpolation functions, that are often used in ITKv4 image registrations. We will also introduce some implementation details of our software in realignment part.

3.2.1. Rigid Body Transforms

Transform is one of the six major parts in ITK registration. It maps moving images to fixed image. In registration, transform can be rigid and nonrigid. A typical rigid transform is affine transform that contains 12 parameters[8] in a homogeneous coordinate system. However, in fMRI pre-processing cases, a simpler transform type: *Rigid Body Transform* is normally used. Rigid body transform uses six parameters to describe motion of a head. Although some of representations of rigid body transform have more than 6 parameters, it has a degree of freedom as 6. Rigid body transform assume heads to be rigid and can only perform translation and rotation movement, each consists of three parameters. In some definitions, reflecting are included

in rigid body transform as well. It excludes scaling, shearing from a full affine linear transform, which is intuitive for a head in scanner, for rigidity of heads.

Matrix-Offset Transform

In ITK, transforms are encapsulated in ITK::Transform. When computing a rigid body transform, or other linear transforms with rotation part, like affine transform, ITK provides *Matrix-Offset Transform* classes.

As we discussed, 3 of the parameters are for translation part in a rigid body transform. They are usually translations in three dimensions. The other 3 parameters is to represent a rotation. Here, we know about the rotation direction and degrees, but we do not know which point the rotation is about, or what is the center of the rotation. Matrix-Offset transforms defines a center. Notice that a center is not a "parameter" of the transform, because it is fixed and could not be updated during optimization (except for those "centered" transform whose center is changing).

The relationship between center, offset and translation is quite confusing. One can define an affine transform by setting matrix and offset, or matrix, center, and translation. When we change a center, the offset is automatically updated according to current matrix and translation. It is recommended to alter only matrix and translation, and set the center fixed.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} M_{00} & M_{01} & M_{02} \\ M_{10} & M_{11} & M_{12} \\ M_{20} & M_{21} & M_{22} \end{bmatrix} \cdot \begin{bmatrix} x - C_x \\ y - C_y \\ z - C_z \end{bmatrix} + \begin{bmatrix} T_x + C_x \\ T_y + C_y \\ T_z + C_z \end{bmatrix} \tag{3.1}$$

Equation 3.1 shows an transform represented in translation and center. Here M is a matrix, C is for center and T is for translation. If we expand the equation and express it in Matrix-Offset way[6], the equation will be:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} M_{00} & M_{01} & M_{02} \\ M_{10} & M_{11} & M_{12} \\ M_{20} & M_{21} & M_{22} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} T_x + C_x - M_{00}C_x - M_{01}C_y - M_{02}C_z \\ T_y + C_y - M_{10}C_x - M_{11}C_y - M_{12}C_z \\ T_z + C_z - M_{20}C_x - M_{21}C_y - M_{22}C_z \end{bmatrix} = M \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} O_x \\ O_y \\ O_z \end{bmatrix} \tag{3.2}$$

In fMRI pre-processing, the center is automatically set to the origin of images, which is predefined in a scanning session. Therefore, images from a same session shares a center. While we are co-register images from multi-modality, we need to specify the center of rotation. We will discuss this in subsection 4.3 *Co-Registration*.

Euler Angles

Here are many representation of a 3-D rotation. *Euler Angles* is one of the most classic ones. Euler angles are named after Leonhard Euler. They are 3 angles around 3 axises in a Cartesian coordinates. Usually noted as α, β and γ, or ϕ, θ and ψ. The convention of Euler angles definition in medical visualization and flight dynamics are quite different. In traditional Euler angle naming system, the three angles are called *attitude*, *bank*, *heading*.[5]

From a conventional Euler angle definition. α and γ are rotation angles around original direction of an axis, and rotated same axis. Notice that although the axis are same, the direction has changed. In figure 3.4, α is angle rotating about z axis, while γ is the angle rotating about Z axis, which is a new axis after two times rotation.

Tait Bryan Angles

Tait-Bryan angles are derivation of Euler angles, named after Peter Guthrie Tait and George H. Bryan. The main difference between them is that Tait Bryan angles rotate about 3 distinct directions,for example, in figure 3.5 x, y and z. While in a classic Euler angles rotation, the rotations is about z, x and z, the first and third rotation are about a same axis.

Tait-Bryan angles are much more popular than classic Euler angles that many application including SPM uses them to represent a 3-D rotation. In our project, we also uses Tait-Bryan angles. In ITK, the Euler angles representation is encapsulated in ITK::Euler3DTransform object. Notice that it is not the classic definition of Euler angle but Tait-Bryan angles. Also notice that in the later 2 rotations, the axises are not the original one before any rotation. Therefore, the order of the rotation is rather important. ITK::Euler3DTranform provides

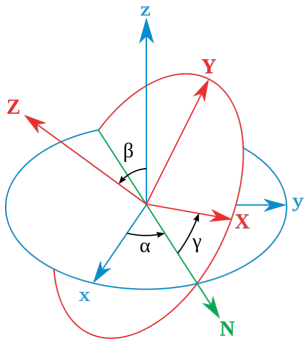

Figure 3.4: Euler Angles, rotated about in z, $N(x')$, z'', from Wikipedia

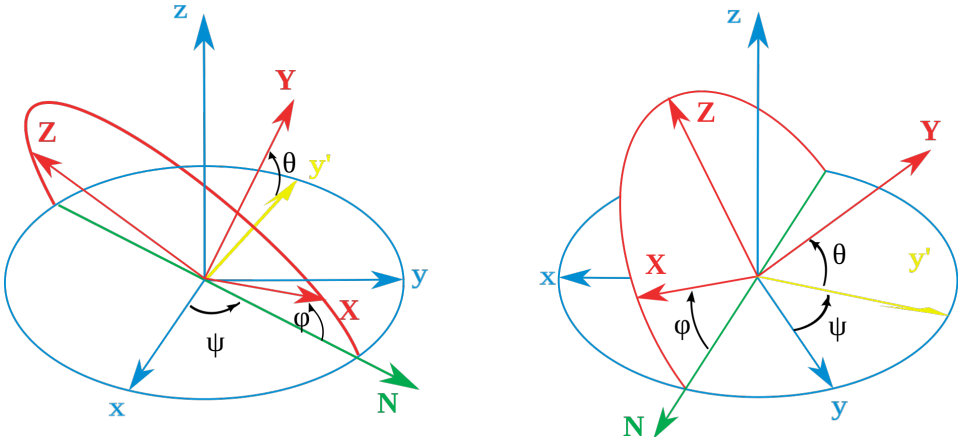

Figure 3.5: Tait Bryan Angles

a member function SetComputeZYX() to set the order between *ZYX* and *ZXY*. Our implementation is in *ZYX* order. Therefore, the representation can be expressed in matrix equation 3.3.[15]

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\psi & sin\psi \\ 0 & -sin\psi & cos\psi \end{bmatrix} \cdot \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix} \cdot \begin{bmatrix} cos\phi & sin\phi & 0 \\ -sin\phi & cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} O_x \\ O_y \\ O_z \end{bmatrix} \tag{3.3}$$

We usually name the 3 angles as *Pitch*, *Roll*, *Yaw* in Tait Bryan angles, instead of Attitude, Bank, Heading. SPM takes this notion as well, see figure 3.6 to find the definition of these notions.

Euler angles representation are categorized into two groups: *Static* and *Dynamic* Euler angles. Static Euler angle describes rotations around axes of the coordinate system, while dynamic Euler angle describes rotations around the axes of the object. In our implementation we are using dynamic Euler angles. When we are using dynamic Euler angle representation of a 3-D rotation, we may encounter the problem of *Gimbal Lock*. Gimbal Lock occurs when the pitch rotation is ±90°, the first rotation and the third become the same. Therefore the object is limited in the vertical plane. The degree of freedom reduce from 3 to 2. However, in fMRI realignment, a pitch angle as large as ±90° is incredible. In normal cases, we dont need to consider the Gimbal Lock problem.

Quaternion and Versor

Another representation of 3-D rotation is *Quaternion*. Despite the degree of freedom in 3-D rotation is 3, quaternion uses 4 parameters: one scalar and a vector of length 3. The basic idea of quaternion is to use one unit rotating axis instead of 3 in Euler angle. Rotation was done in one time, not around one of x, y or z axis but a composite unit axis.

ITK defines ITK::QuaternionRigidTransform for us. The parameters list contains 7 parameters. The first 4 are the components of the quaternion representing a 3-D rotation. And the other 3 parameters defines trans-

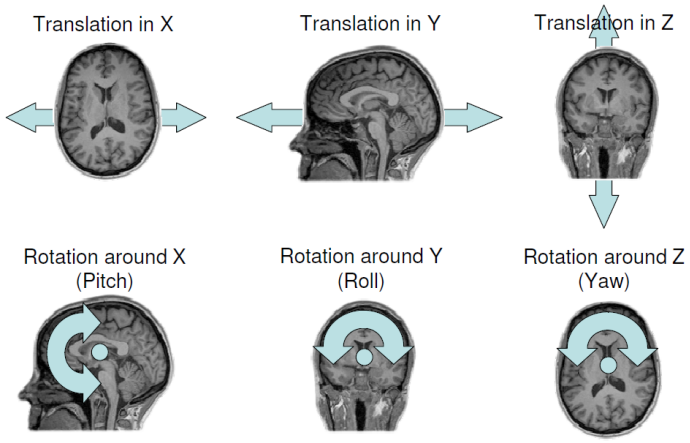

Figure 3.6: Representaion of Rigid Body Transform, in Tait Bryan Angles, Image from *Matthew Brett and Rik Henson*

lation in each dimension.

Versor is a normalized quaternion. We notate the scalar as q_0, which stands for the rotation angle around the composite axis. q_0 is computed from vector $v = \{q_1, q_2, q_3\}$ by $q_0^2+q_1^2+q_2^2+q_3^2 = 1$. ITK provides ITK::VersorRigid3DTransform to present a 3-D rigid body transform. There are 6 parameters in ITK::VersorRigid3DTransform. The first 3 are the components of versor vector. Notice that the vector is no longer a unit vector, as it was in ITK::QuaternionRigidTransform. The conversion from quaternion or versor to 3 by 3 rotation matrix is shown in 3.4.

$$M = \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2-q_0q_3) & 2(q_0q_2+q_1q_3) \\ 2(q_1q_2+q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_0q_1+q_2q_3) & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix} \tag{3.4}$$

In NIfTI-1 image data format, the spatial information are store in 2 method. One is a 3 by 4 rotation matrix. Another is a versor vector with a translation vector. According to convention, the new spatial information after realignments is re-written and stored in the 3 by 4 rotation matrix. The code of qform_code should be set to 2 which stands for *Coregistered*. While the versor parameters doesn't change. The code of sform_code is kept as 1, which demonstrates that the spatial information are from the scanner originally.

Rotation Matrix

An affine transform can be expressed in a 3 by 4 matrix. Rigid body transform is a special case of affine transform can surely be presented in a 3 by 4 matrix as well. A 3 by 4 matrix representation uses 12 parameters instead of 6. But it makes computation much easier. In our implementation, we use ITK::Euler3DTransform in optimization. Then we convert the Tait Bryan angles into 3 by 4 matrix, to compute the new spatial matrix after realignment and re-write it into NIfTI header. We will see details in 4.2.4.

3.2.2. Similarity Metrics

Metric is another important part in ITKv4 registration. They are objects evaluating how similar two objects are. When used in image registration, it measures how well the two images matches. The method to compute a metric is various. And ITK also implemented most of them. In ITKv4 registration, metrics are computed in physical space, that is the virtual image domain.

In our implementation of realignment, we adopt ITK::MeanSquareImageToImageMetricv4 and ITK::MattesMutualInformationImageToImageMetricv4. Optimizers depends on different metric types.

Mean Square Metrics

In intensity-based image registration, ITK::MeanSquareImageToImageMetricv4 is the most commonly used one. The metric is defiend as *Mean Square Error* (*MSE*), as 3.5

$$MS(F,M) = \sum_{i \in ROI} (F_i - M_i)^2 \tag{3.5}$$

However, mean square error as metric is sensitive to noise and changing in brightness. In multi-modality registration cases, the metrics works badly. We appeal to mutual information metric instead. See 4.3 Co-registration.

Normalized Cross Correlation

Another common used similarity metric in image registration is *Normalized Cross Correlation.* It computes pixel-wise cross correlation. To avoid effect from lighting conditions, a normalized version of cross correlation is more used. See 3.6.

$$NC(F,M) = -\frac{\sum_{i=1}^{N}(F_i \cdot M_i)}{\sqrt{\sum_{i=1}^{N} A_i^2 \cdot \sum_{i=1}^{N} B_i^2}} \tag{3.6}$$

In ITK, ITK::NormalizedCorrelationImageToImageMetric is provided for intensity-based image registration cases. Notice that this class does not have a ITKv4 version in recent ITK releases. That is the metric is computed on grid of fixed image domain, not virtual image domain.

3.2.3. Optimizers

ITK::Optimizer are objects for optimization. It warps optimization algorithms from vnl_optimizers. An optimizer takes a cost function or a metric object as input. And then iteratively find the minimum value of the cost function or metric when a stopping criterion is satisfied. There are many optimization algorithms integrated into ITK. Some of them return a single value, as defined in ITK::SingleValuedNonLinearOptimizer. On the opposite, some optimizers return multiple values are defined in ITK::MultipleValuedNonLinearOptimizer. See 4.1.

In the project, we not only developed a software for fMRI analysis, but also did a benchmarking on different single value returned optimizers in realignment. The experiment of benchmark are discussed in more detail in chapter 5. Here we briefly introduces the theoretical basis of these optimization algorithms.

Regular Step Gradient Descent Optimizer

ITK::RegularStepGradientDescentOptimizerv4 is a variant of gradient descent optimizers, which attempts to prevent it from taking steps that are too large. The method finds local optima.

The idea is not complex. Assuming we are driving on the direction of metric derivate descent, $-\nabla M(p)$, at a speed step length of γ. Step length is set to 1 by default, which can be changed by setting the learning rate. However, when the direction of metric derivative descent changes dramatically, we supposed that we have passed a local optima. That is the local optima is near where we are. It is reasonable to slow down our speed and search. The current step length is $\lambda^m \gamma_0$ after we have passed a local optima m times. See 3.7.

$$p_{n+1} = p_n - \lambda^m \gamma_0 \nabla M(p_n) \tag{3.7}$$

Gradient Descent Line Search Optimizer

Line search in optimization is method to update step length or learning rate in every iteration. The step length can be updated either in an exact way or in an inexact way. ITK::GradientDescentLineSearchOptimizerv4 is also a variant of simple gradient descent line search optimizer. However, it update the learning rate by computing a *Golden Section Line Search.*

$$p_{n+1} = p_n - \gamma_n \nabla M(p_n) \tag{3.8}$$

Where γ_n is chosen to satisfy:

$$\gamma_n = \arg\min_{\gamma_n} M(p_n - \gamma_n \nabla M(p_n)) \tag{3.9}$$

A good guess of learning rate γ_n that satisfying golden section. The idea of golden section line search is based on golden ratio. For example in our recent three iterations, we have metric value $M(p_1), M(p_2), M(p_3)$,

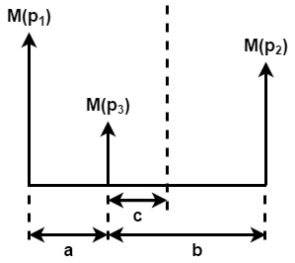

Figure 3.7: Golden Section Line Search

as shown in 3.7. It is apparent that there is a local optima between p_1 and p_2. A golden section search finds a learning rate γ that makes p_4 at the golden section point, satisfying $c : a = a : b$, which is thought to be most efficient in optima searching.

$$\frac{b}{a} = \frac{1+\sqrt{5}}{2} \approx 1.618 \tag{3.10}$$

The upper and lower limit below determine the range of values over which the learning rate can be adjusted by the golden section line search. While ϵ determines the accuracy of the line search. A smaller ϵ leads to more accurate optimization, but costs more time.

Conjugate Gradient Line Search Optimizer

Similar as ITK::GradientDescentLineSearchOptimizerv4, ITK::ConjugateGradientLineSearchOptimierv4 also update learning rate according to golden section line search. However, in this case the gradient $-\nabla M(p)$ is replaced by *Polak-Ribière Conjugate Gradient*, see 3.11, d is for the conjugate gradient. Notice that learning rate γ_n is still computed from golden section line search.

$$p_{n+1} = p_n + \gamma_n s_n \tag{3.11}$$

Suppose we are using Mattes mutual information as our metric. The metric is twice differentiable at its minimum. Similar as in a normal gradient descent method, the first step goes in a direction of negative gradient. However, the direction changes in following iterations. For example, in n^{th} iteration, we goes in a conjugate direction s_n, instead of $-\nabla M(p_n)$. The conjugate gradient is computed from 3.12.

$$\begin{cases} s_0 = -\nabla M(p_0) \\ s_n = -\nabla M(p_n) + \beta s_{n-1} \end{cases} \tag{3.12}$$

From 3.8 we can see that conjugate gradient method is much faster than traditional gradient descent method. The conjugate gradient method prevents the zigzag trace, which makes the optimization more efficient.

There are many method to compute β, Polak-Ribière is one of them. In ITKv4, Polak–Ribière formula is adopted. See 3.1.

Fletcher–Reeves	$\beta_n^{FR} = \frac{\triangle p_n^T \triangle p_n}{\triangle p_{n-1}^T \triangle p_{n-1}}$
Polak–Ribière	$\beta_n^{PR} = \frac{\triangle p_n^T (\triangle p_n - \triangle p_{n-1})}{\triangle p_{n-1}^T \triangle p_{n-1}}$
Hestenes-Stiefel	$\beta_n^{HS} = \frac{\triangle p_n^T (\triangle p_n - \triangle p_{n-1})}{s_{n-1}^T (\triangle p_n - \triangle p_{n-1})}$
Dai–Yuan	$\beta_n^{DY} = \frac{\triangle p_n^T \triangle p_n}{s_{n-1}^T (\triangle p_n - \triangle p_{n-1})}$

Table 3.1: Formulas to compute β

Quasi Newton Optimizer

Quasi-Newton method is variant of well-known *Newton-Raphson* method, which is used to find roots or ex-

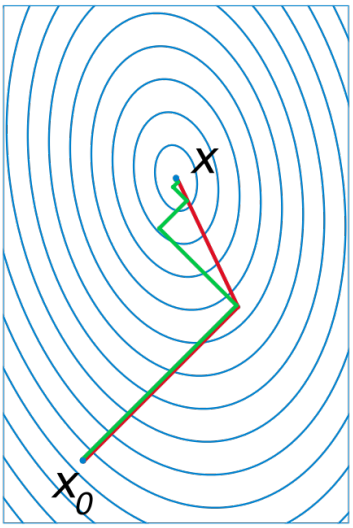

Figure 3.8: Gradient Descent Linear Search Method & Conjugate Gradient Descent Line Search Method

treme values of a function[13]. like 1^{st} order optimization in gradient descent, Newton's method takes 2^{nd} order derivatives. Therefore Newton's method takes more time and gives more accurate result. In Newton-Raphson method, we expand the metric function into second order Taylor series as 3.13. Here H is our Hessian matrix.

$$M(p_{n+1}) \approx M(p_n) + \triangle p_n^T \nabla M(p(n)) + \frac{1}{2} \triangle p^T H \triangle p \tag{3.13}$$

Hessian matrix is a square matrix consists of partial derivatives of a single valued function, see 3.14.

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial p_1^2} & \frac{\partial^2 f}{\partial p_1 \partial p_2} & \cdots & \frac{\partial^2 f}{\partial p_1 \partial p_n} \\ \frac{\partial^2 f}{\partial p_2 \partial p_1} & \frac{\partial^2 f}{\partial p_2^2} & \cdots & \frac{\partial^2 f}{\partial p_2 \partial p_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial p_n \partial p_1} & \frac{\partial^2 f}{\partial p_n \partial p_2} & \cdots & \frac{\partial^2 f}{\partial p_n^2} \end{bmatrix} \tag{3.14}$$

Take gradient of this approximation with respect to $\triangle p$ and set it to zero, we get 3.15. α is a parameter satisfying *Wolfe Condition*.

$$\begin{cases} p_{n+1} = p_n - \alpha_n H_n^{-1} \nabla M(p_n) \\ y_n = \nabla M(p_{n+1}) - \nabla M(p_n) \end{cases} \tag{3.15}$$

The full Newton-Raphson method requires Hessian matrix for optima searching. However, it is time costing and or unfeasible to compute Hessian matrix at every iteration. Quasi-Newton method provided a solution to estimate the matrices in an iterative way. In 3.15, y_n is used to estimate Hessian matrix at next iteration H_{n+1} and its inverse H_{n+1}^{-1} using *Sherman-Morrison* formula[19].

- DFP (Davidon–Fletcher–Powell Formula):
  $H_{n+1} = (I - \frac{y_n \triangle p_n^T}{y_n^T \triangle p_n}) H_n (I - \frac{\triangle p_n y_n^T}{y_n^T \triangle p_n}) + \frac{y_n y_n^T}{y_n^T \triangle p_n}$
  $H_{n+1}^{-1} = H_n^{-1} + \frac{\triangle p_n \triangle p_n^T}{\triangle p_n^T y_n} - \frac{H_n^{-1} y_n y_n^T H_n^{-1}}{y_n^T H_n^{-1} y_n}$
- BFGS (Broyden–Fletcher–Goldfarb–Shanno Algorithm):
  $H_{n+1} = H_n + \frac{y_n y_n^T}{y_n^T \triangle p_n} - \frac{H_n \triangle p_n (H_n \triangle p_n)^T}{\triangle p_n^T H_n \triangle p_n}$
  $H_{n+1}^{-1} = (I - \frac{\triangle p_n y_n^T}{y_n^T \triangle p_n}) H_n^{-1} (I - \frac{y_n \triangle p_n^T}{y_n^T \triangle p_n}) + \frac{\triangle p_n \triangle p y_n^T}{y_n^T \triangle p_n}$

- Broyden's Method:
  $H_{n+1} = H_n + \frac{y_n - H_n \triangle p_n}{\triangle p_n^T \triangle p_n} \triangle p_n^T$
  $H_{n+1}^{-1} = H_n^{-1} + \frac{(\triangle p_n - H_n y_n) \triangle p_n^T H_n^{-1}}{\triangle p_n^T H_n y_n}$
- SR1 (Symmetric rank-one):
  $H_{n+1} = H_n + \frac{(y_n - H_n \triangle p_n)(y_n - H_n \triangle p_n)^T}{(y_n - H_n \triangle p_n)^T \triangle p_n}$
  $H_{n+1}^{-1} = H_n^{-1} + \frac{(\triangle p_n - H_n^{-1} y_n)(\triangle p_n - H_n^{-1} y_n)^T}{(\triangle p_n - H_n^{-1} y_n)^T y_n}$

ITK::QuasiNewtonOptimizerv4 implements the algorithm with BFGS Hessian estimation. Notice that ITK::QuasiNewtonOptimizerv4 does not support setting parameter scales manually. We used ITK::RegistrationParameterScalesFromPhysicalShift to estimate parameter scales from metric function.

LBFGS Optimizer

LBFGS is short for *Limited-Memory Broyden–Fletcher–Goldfarb–Shannon Algorithm.* It is developed from Quasi-Newton optimization method, estimating inverse Hessian matrix using limited computer memory. A normal BFGS estimation of Hessian matrix takes too much resources in computation, especially when the number of parameters is very large.

Consider 3.15 in Quasi-Newton optimizer, we need to compute $-H_n^{-1} \triangledown M(p_n)$ in every iteration. In Quasi-Newton optimizer it was stored in a $n \times n$ matrix, and it was computed from H_{n-1}^{-1} from last iteration. However, in LBFGS, we computed them from last m iterations. Here we defined $H_n^{-1(0)}$, as initial approximate inverse Hessian matrix at n^{th} iteration.

$$\begin{cases} \triangledown p_n = p_{n+1} - p_n \\ y_n = \triangledown M(p_{n+1}) - \triangledown M(p_n) \\ \rho_n = \dfrac{1}{y_n^T \triangle p_n} \\ H_n^{-1(0)} = \dfrac{y_{n-1} \triangle p_{n-1}^T}{y_{n-1}^T y_{n-1}} \end{cases} \tag{3.16}$$

Then we start to compute $-H_n^{-1} \triangledown M(p_n)$ from last m iterations. At first, we copy $\triangledown M(p_n)$ into a variable q_n. We loop from $i = n-1$ down to $n-m$, to update q_n by 3.17.

$$\begin{cases} \alpha_i = \rho_i \triangle p_i^T q_n^{old} \\ q_n^{new} = q_n^{old} - \alpha_i y_i \end{cases} \tag{3.17}$$

After we get updated q_n in the beginning of every iteration, we store $H_n^{-1(0)} q_n$ into a variable noted as z_n. Then we start another loop from $i = n-m$ up to $n-1$, to update z_n, see 3.18.

$$\begin{cases} \beta_i = \rho_i y_i^T z_n^{old} \\ z_n^{new} = z_n^{old} + \triangle p_i (\alpha_i - \beta_i) \end{cases} \tag{3.18}$$

Up to mow we have final z_n in every iteration, this z_n is estimation of $H_n^{-1} \triangledown M(p_n)$. That is, in LBFGS, we do not compute the multiplication of an $n \times n$ matrix and a gradient vector. We estimate the product of them directly, using information from m past iterations instead 1. Above method is what called *Two Loop Recursion.*

ITK implements LBFGS algorithm in ITK::LBFGSOptimizerv4. And the memory size m is set to 5 by default.

LBFGSB Optimizer

LBFGSB is for Limited-BFGS-Bounded, which is a variant of LBFGS optimizers, whose parameters are limited in upper and lower boundaries. We could add upper and lower boundaries to each of the parameters. It was implemented in ITK::LBFGSBOptimizerv4. Notice that these two optimizers are variant of Quasi-Newton optimizer, thus we can not set the parameter scales manually.

$$l_i \leq p_i \leq u_i \tag{3.19}$$

In ITK::LBFGSBOptimizerv4 we are able to set how we apply the boundaries. There are 4 options: bounded lower limit, bounded upper limit, bounded in both and unbounded.

Amoeba Optimizer

Amoeba optimization is also called *Nelder–Meade* method or *Downhill Simplex* method, which is a heuristic method that does not compute derivatives of metric function. *Simplex* is an important concept in Amoeba algorithm. Simplex is a polytope with $(n+1)$ vertices in n dimensional space. It changes its shape and goes down a hill and reach the basin of a function, that is why this method named Amoeba. In our case we have 6 parameters for rigid body transform, thus we have a polytope with 7 vertices.

There four basic process on the simplex and its vertices: *Reflection*, *Expansion*, *Contraction*, *Shrink*. The pipeline of Amoeba algorithm is shown in 3.9. P_c, P_r, P_e represents replacing p_7 by results from contraction, reflection and expansion respectively.

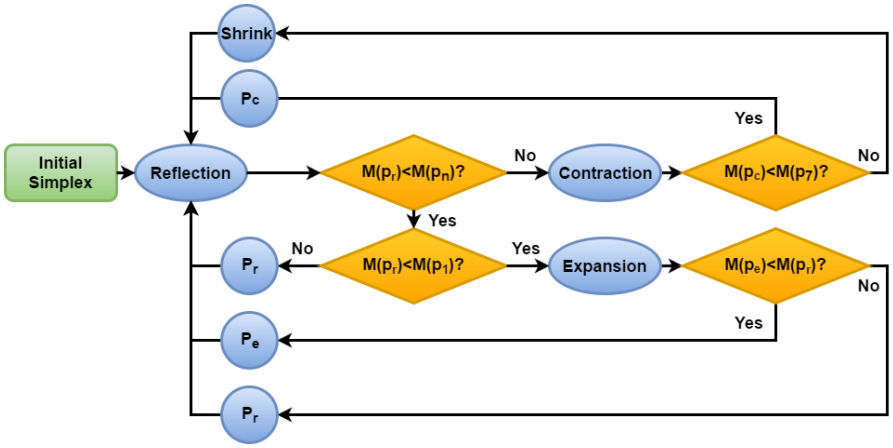

Figure 3.9: Pipeline of Downhill Simplex Method

Reflection: Order the vertices according to metric value at the point. We have: $M(p_1) < M(p_1) < \cdots < M(p_7)$. Here p_7 is the worst vertex in the simplex, which needs to be replaced. The replacing candidate is computed as reflection of p_7, about centroid of p_1 to p_6.

$$p_{reflection} = p_{centroid} + \alpha(p_{centroid} - p_7)(\alpha > 0) \tag{3.20}$$

Expansion: If the new point we get from reflection is even better than the best point p_1, we will compute the expanded point as 3.21. We replace the worst point by the expanded one and then start another reflection.

$$p_{expanded} = p_{centroid} + \gamma(p_{reflection} - p_{centroid})(\gamma > 0) \tag{3.21}$$

Contraction: If the new point we get from reflection is still worse than the less worst point p_6, we compute the contracted point as 3.22.

$$p_{contracted} = p_{centroid} + \rho(p_7 - p_{centroid})(0.5 > \rho > 0) \tag{3.22}$$

Shrink: In the worst case, if the point after contraction is even worse than the original worst point. We need to shrink the simplex as 3.23.

$$p_i = p_1 + \sigma(p_i - p_1)(i = 2,3,\cdots,7) \tag{3.23}$$

α, γ, ρ, σ are reflection, expansion, contraction and shrink coefficients, which are set to 1, 2, $\frac{1}{2}$, $\frac{1}{2}$ by default respectively. Notice that if an initial simplex is too small in size, it may be easier stuck in local optima. Amoeba algorithm does not compute derivatives of a metric function. It is a heuristic search method, which

means it can converge to non-stationary points.

ITK::AmoebaOptimizerv4 implements Nelder–Meade algorithm. In our implementation, we generated the initial simplex automatically. Parameter convergence tolerance Function convergence tolerance are two stopping criterions. The first terminates iteration when the diameter of simplex is smaller than a value. The latter terminates iteration when the difference in metric function is smaller than a value.

One Plus One Evolutionary Optimizer

ITK::OnePlusOneEvolutionaryOptimizerv4 searches for global optima in a very intuitive way. The terms come from Darwin's evolution theory. We search randomly for a mutation point. If the mutation makes the cost function smaller, we keep it. If not, we search for another mutation point. Like this, we update our current position and radius along iterations, finally stop at an optima. Figure 3.10 shows an example of how one plus one evolutionary optimizer works[20].

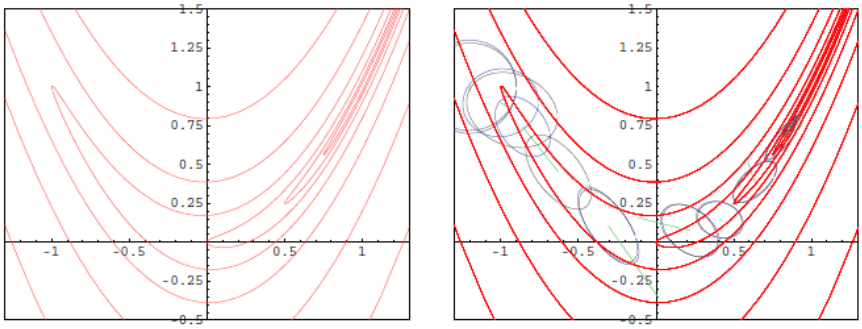

Figure 3.10: Illustration of 1 + 1-ES algorithm on the Rosenbrock function, from *Martin Styner*

The searching process start with a position p_0 in parameter space, and initial radius r_0. Then we select a random point around p_0. The selection was according to an isotropic probability distribution function (PDF) centered at p_0. The PDF in this case is a Normal Ditribution, generated by ITK::NormalVariateGenerator, based on the searching radius. Every time we find a new position, we may change our searching radius, by grow factor c_{grow} and shrink factor c_{shrink}, see 3.24.

$$r_{n+1} = \begin{cases} r_n \cdot c_{grow}, & (M(p_{n+1}) < M(p_n)) \\ r_n \cdot c_{shrink}, & (M(p_{n+1}) \geq M(p_n)) \end{cases} \tag{3.24}$$

According to *Hans-Paul Schwefel*, the best c_{grow} value is between 1.0 and 1.1. And the best shrink factor can be computer from 3.25. The process will stop when the stopping criterion is met. One is the number of iterations exceeds a limitation. Another is that the shrinked radius is once too small compare to ϵ that we defined.

$$c_{shrink} = c_{grow}^{-\frac{1}{4}} \tag{3.25}$$

3.2.4. Interpolation

Interpolation is the last part in ITKv4 registration framework. We use interpolators to compute voxel values at non-grid position. For image object in ITK, there are 4 types of interpolators: *Nearest Neighbor Interpolator*, *Linear Interpolator*, *B-Spline Interpolator*, *Windowed Sinc Interpolator*. Users may define their custom interpolation functions as well. ITK also provides interpolators for ITK::VectorImages, which is beyond the boundary of our discussion.

Figure 3.11 is from Michael Unser, 2002, that compares the performance-cost ratio of different interpolation algorithms. The result comes from a series of rotation in 15 times, each time 24 degrees. We see that there is not an optimal interpolateor. Only a best suited one exists. However, B-Spline interpolators, generally performs better than others.

Nearest Neighbor Interpolator

Nearest neighbor interpolator is the most simple interpolation function. In ITK, it was implemented in ITK::NearestNeighborInterpolateImageFunction, which copies the intensity of the nearest neighbor to a non-grid position.

Linear Interpolator

Linear interpolator assumes the intensity at a non-grid position is decided only by 2^n neighbor points, where n is the dimension. It compute the intensity as a linear combination of those points, based on distance to each point. In ITK, linear interpolators are implemented in ITK::LinearInterpolateImageFunction. Unlike other scalar interpolators, ITK::LinearInterpolateImageFunction works for ITK::VectorImage as well.

B-Spline Interpolator

The term B-spline is short for basis spline, named by Isaac Jacob Schoenberg. It uses a polynomial function to interpolate intensities of voxels. We take 1-D case as an example. The interpolation function of B-spline with n control points, of order k, is defined as 3.26.

$$I(t) = \sum_{i=0}^{n} N_{i,k}(t) I_i \tag{3.26}$$

Here $N_{i,k}(t)$ is what we called *Basis Function*. It acts as weighting coefficients of intensities. The weighting coefficient can be computed from *Cox - de Boor Recursive Formula*, as shown in 3.27. $N_{i,0}(t)$ is the zero-order basis function, which is defined as 1 on interval $[t_i, t_{i+1})$. According to the formula, we may find that basis function $N_{i,k}(t)$ is non-zero if t locates in interval $[t_i, t_{i+k+1})$. That is, in k-order interpolation, $k+1$ voxels contribute to the interpolation.

$$N_{i,k}(t) = \frac{t - t_i}{t_{i+k} - t_i} N_{i,k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} N_{i+1,k-1}(t) \tag{3.27}$$

In ITK, B-spline interpolator is implemented in ITK::BSplineInterpolateImageFunction, in which we can set the order from 0 to 5. In zero-order case, the interpolator becomes a floor interpolator. When order equals 1, it becomes linear interpolator. Notice that the order of interpolator should be set before setting an image.

Windowed Sinc Interpolator

As we know, the *sinc* function ($sinc(t) = \frac{sin(\pi t)}{\pi t}$) is the inverse Fourier transform of a ideal low-pass window filter in frequency domain. Therefore, *sinc* function is a perfect interpolator if Nyquist frequency is satisfied. However ,the support of *sinc* function is infinite. Windowed *sinc* function is to approximate *sinc* function in a certain range of support to interpolate data.

There are many instances of windowed *sinc* function, depending on different window functions. For example, Cosinus window, Welch window, Lancos window, Hamming window etc.. The interpolating kernel is windowed *sinc* function, see 3.28.

$$I(x,y) = \sum_{i=\lfloor x \rfloor + 1 - m}^{\lfloor x \rfloor + m} \sum_{j=\lfloor y \rfloor + 1 - m}^{\lfloor y \rfloor + m} I_{j,k} K(x-i) K(y-j) \tag{3.28}$$

ITK::WindowedSincInterpolateImageFunction, by Paul A. Yushkevich, uses windowed *sinc* function to interpolate. We are able to set the kernel radius (m in 3.28) and the window function. ITK provides some useful window functions, for example Function::HammingWindowFunction, Function::LanczosWindowFunction, Function::CosineWindowFunction, Function::WelchWindowFunction and Function::BlackmanWindowFunction.

3.2.5. Implementation Details

In our implementation of FIONA, we select Euler angle as our transform prototype, linear interpolator as our interpolator. As for metric and optimizer, we did a benchmark on them of 10 experiments. From the result, we decided to apply quasi-Newton optimizer and mean square metric for realignment.

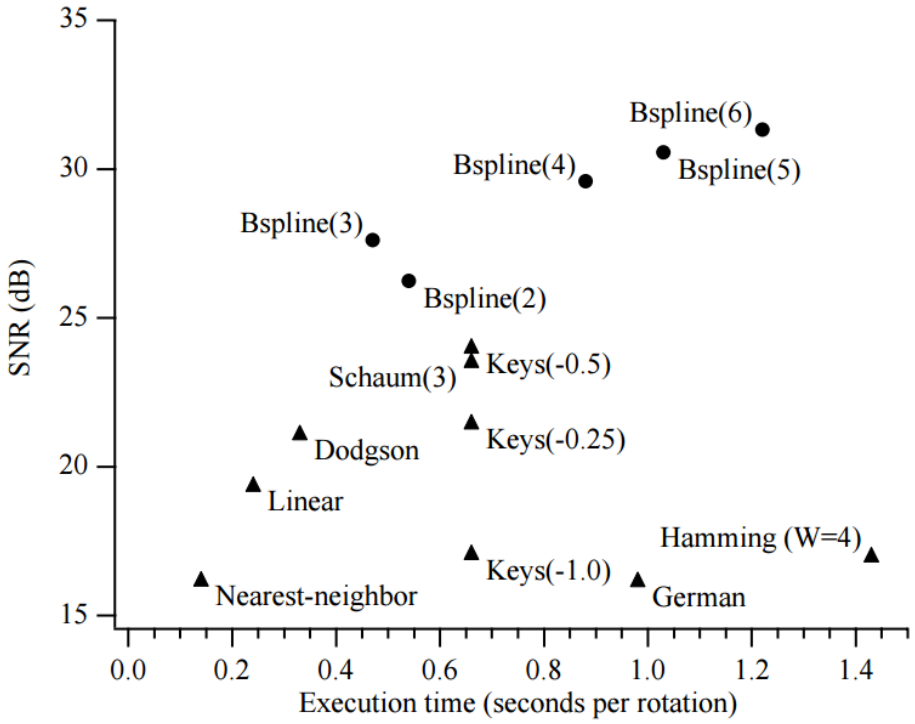

Figure 3.11: Performance-Cost Comparison of Different Interpolation Algorithms, from Michael Unser, *Splines: a perfect fit for medical imaging*

After realignment, we have parameters list of six column: *Pitch, Roll, Yaw, Translation in X, Translation in Y, Translation in Z*. We have two options to apply the realigned parameters to moving images.

Re-Write NIfTI Header

Similar as functionality of "Estimate Only" in SPM, we can choose to only change the NIfTI header of a moving image, without changing the image data. NIfTI is short for *Neuroimaging Informatics Technology Initiative*. NIfTI-1 is a data format storing neuroimaging data (see http://nifti.nimh.nih.gov/). The suffix of NIfTI files are .hdr or .img for compressed 4-D images, and .nii for single 3-D images.

As we introduced before, NIfTI files has a header of 348 bytes. It contains all the metadata about the image, including transforms from voxel index (*i, j, k*) to spatial location (*x, y, z*). In this case, We only want to change the fields that store this spatial transform, as well as a NIfTI form_code identifying the status of spatial information of an image. We planned to perform the changing header task using ITK::ChangeInformationImageFilter. However, some fields in the header will be changed by using ITK::NiftiImageIO. Thus, we preferred to write NIfTI files manually.

As in convention, we keep qform_code as 1, standing for "Scanner Position" not changing. Also we keep all the fields in method 2, which uses a quaternion to store the transform. The things to to is listed here:

- Change sform_code from 1 to 2, which stand for "Coregistered".
- Re-Write the 3 by 4 matrix in method 3, which is stored in srow_x[4], srow_y[4], srow_z[4].

We compute the new transform matrix by a matrix multiplication in 4 by 4. We can export transform matrix from ITK::Euler3DTransform directly. Notice that ITK uses LPS (Left, Posterior, Superior) coordinate system, while NIfTI uses RAS (Right, Anterior, Superior) coordinate system. Therefore, we need to negate two parameters before computation.

Also notice that there is a extender block of 4 bytes in NIfTI files, between the header and image data. Do

not forget to allocate memories for them.

Re-Slice Moving Image

In most fMRI researches, only re-write headers of NIfTI files is not sufficient. We need to make images in a same session share a same coordinate grid. Similar as "Estimate & Reslice" functionality in SPM, our project also provide option to re-slice moving images.

In our implementation, the re-slice process is done by using ITK::ResampleImageFilter. We set the fixed image as reference image, set the ITK::Euler3DTransform as transform, and set ITK::LinearInterpolateImageFunction as an interpolator. The spatial informations, like spacing, origin, orientation and region of fixed image are copied to moving images. The re-sliced images are basically ready for coming statistical analysis.

3.3. Co-Registration

Co-registration is a special case of registration, of images scanned from multi-modalities. Often we need to match the structural data and the functional data, to obtain a better localization of activated area, and a better visualization as well. Structural data are often a T1 weighted MR image, which was taken in the same session of functional data.

The main problem in co-registration is that spatial resolution of structural data is much higher than functional data. Because functional MR scans trade-off spatial resolution for a better temporal resolution. A TR of functional MR image is about $2s$, while a typical T1 image takes about 8 or 10 minutes.

In our implementation, we set functional image as fixed image, and anatomical image as moving image. Because all the computation is done in virtual image domain Therefore, the spatially low resolution in fixed image doesn't matter. The framework of our co-registration functionality is shown in 3.12.

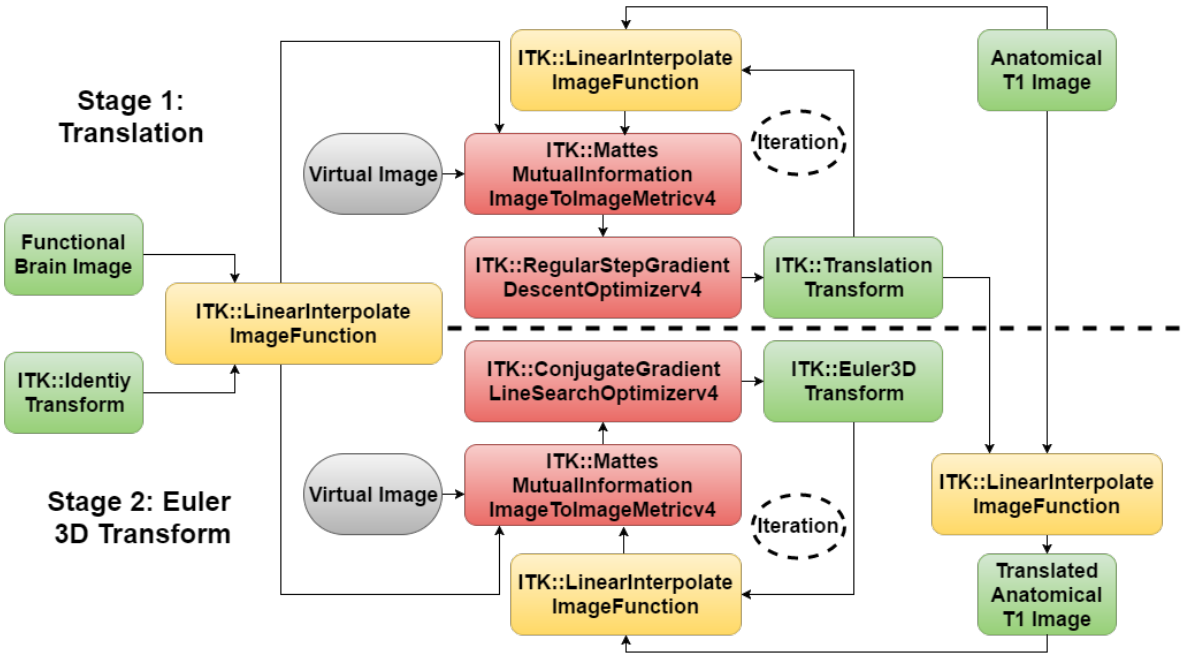

Figure 3.12: Multi-Stage Co-Registration Framework

In the framework, we applied some specific registration techniques. First, we used Mattes mutual information as similarity metric, for multi-modality case. Second, we used multi-stage registration for large movement and centers apart distantly. Third, we used multi-resolution approach to speed up registration and avoid local optima. In following subsections, we will illustrate them.

3.3.1. Multi-Stage Registration

The basic idea of multi-stage registration is first applying a coarse simple transform, to make the two images closer. Then apply a finer complex transform to get accurate transform parameters. That is, there are multiple registrations. Each has its own different transform, metric, optimizer and interpolator.

ITKv4 implements multi-stage registration by defining a class ITK::CompositeTransform. A composite transform is an object consist of multiple transforms, in an order. It acts as applying each component transform one by one. In our experiment, the original distance between centers of T1 image and of function image is very large. We first need to apply a ITK::TranslationTransform, to match the two centers. And then we apply a ITK::Euler3DTransform, as we did in realignment.

3.3.2. Mattes Mutual Information Metrics

In multi-modality registration, metrics we introduced in realignment, mean square metric and cross correlation metric are not working well. However, mutual information works. Thus, we appeal to Mattes mutual information as our metric. In Shannon information theory, mutual information is defined as a measure of mutual dependency between two image intensities. The dependency is not defined explicitly. Therefore we can use mutual information as metric in multi-modality registration.

Mattes mutual information, named after D. Mattes, is defined in 3.29[14]. $p(\iota,\kappa|\mu)$ is the joint PDF (probability distribution function), $p_M(\iota|\mu)$ is marginal PDF of moving image and $p_F(\kappa)$ is marginal PDF of fixed image. μ is the transform parameter.

$$S(\mu) = -\sum\sum p(\iota,\kappa|\mu) log(\frac{p(\iota,\kappa|\mu)}{p_T(\iota|\mu)p_R(\kappa)}) \tag{3.29}$$

Mattes estimates the PDFs using B-spline Parzen window method, shown in 3.30. In the joint PDF, α is a normalization factor which makes $\sum p(\iota,\kappa) = 1$.

$$\begin{cases} p(\iota,\kappa|\mu) = \alpha \sum\limits_{x\in V} \beta^{(0)}(\kappa - \dfrac{f_F(x) - f_F^\circ}{\Delta b_F}) \times \beta^{(3)}(\iota - \dfrac{f_M(f(x|\mu)) - f_M^\circ}{\Delta b_T}) \\ p_M(\iota|\mu) = \sum\limits_{\kappa} p(\iota,\kappa|\mu) \\ p_F(\kappa) = \alpha \sum\limits_{x\in V} \beta^{(0)}(\kappa - \dfrac{f_F(x) - f_F^\circ}{\Delta b_F}) \end{cases} \tag{3.30}$$

$\beta^{(3)}(x)$ is a cubic spline Parzen window defined in 3.31. $\beta^{(0)}(x)$ is a zero-order spline Parzen window. $\beta^{(0)}$ is defined as a unit pulse centered at x. f_F° and f_M° are minimum intensity values in fixed and moving images. Δb_F and Δb_M are intensity ranges of each bin in fixed and moving images.

$$\beta^{(3)}(x) = \begin{cases} \dfrac{1}{6}(4 - 6x^2 + 3|x|^3) & 0 \le |x| < 1 \\ \dfrac{1}{6}(2 - |x|)^3 & 1 \le |x| < 2 \\ 0 & 2 \le |x| \end{cases} \tag{3.31}$$

ITK provides ITK::MattesMutualInformationImageToImageMetricv4. In our implementation, we used ITK::MattesMutualInformationImageToImageMetricv4 as our similarity metric in co-registration case. We are able to set the bins in Parzen window estimation.

3.3.3. Multi-Resolution Approach

Multi-resolution approach will help a registration in speed, accuracy and robustness. The basic idea is first to perform a registration process in a lower resolution image, with fewer voxels and more smoothing. After a lower level registration, we start from the parameters we get as initial parameters in a higher level registration[16]. The repetition ends when the highest resolution registration is finished. This is often called

a *Coarse to Fine Strategy*.

ITK implements multi-resolution registration in ITK::ImageRegistrationMethodv4. We set a multi-resolution pyramid by setting levels, and shrink factors or smoothing variance in each level. A figure of ITK multi-resolution registration is shown in 3.13.

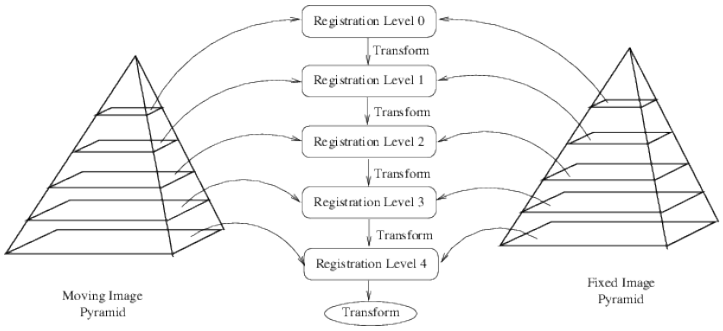

Figure 3.13: Multi-Resolution Registration Framework in ITK

3.4. Normalization

Human brains are very different from each other. We need to locate a task-related activation area onto an anatomical cortex image. We surely can do this via co-registration in every single case. However, it is quite time consuming and needs certain neuroanatomic skill. Another option is to register all the anatomical images into one standard brain or an atlas. Normalization is a process, to register structural images to structural image.

The most widely used atlas is the one called Talairach atlas, which is come from an old French lady. Another most used atla is the MNI (Montreal Neurological Institute) one, which is generated from average of 152 different brains. Neuroimages can be converted between Talairach and MNI atlas.

Notice that the registration transform prototype is more complex than rigid body transform, that used in realignment and co-registration. The transform in normalization can have non-linear part, to perform deformable shape transform. In most normalization algorithms, they adopted multi-stage registration as well. The first step is a linear part of the composite transform. The second part is non-linear.

For the linear transform, affine transform is adopted instead of simple rigid body transform. Because we need scaling, shearing between two head structures. For the nonlinear part, the transform is defined differently in different region, as the image is an elastic body. A typical non-linear registration may have thousands of parameters in total.

In our project, we did not implement normalization. Because our project is designed for fMRI processing before TMS Neural Navigator (NeNa). NeNa performs point-based registration between lab-space measurements and MRI space of craniotopic landmarks, which requires the imaged head and real head to be registerable with rigid body registration. Normalization deforms the MRI image, which precludes such a rigid body registration.

3.5. Spatial Smoothing

Spatial smoothing assigns intensity at each voxel by a weighted average of neighboring BOLD response. We used a traditional Gaussian smoother in our implementation.

We appeal to ITK::DiscreteGaussianImageFilter to perform a spatial smoothing. The smoothing process is done optionally together with re-slice process. Notice that in "Estimate Only" mode, we can not apply spatial

smoothing, because image data are not touched in that mode.

3.6. Temporal Filtering

Spatial filtering smoothes data in each TR volume. On the contrary, temporal filtering smoothes voxels at same spatial position by volumes in neighboring TRs. That is, spatial smoothing is done in 3 dimensions while temporal smoothing is done in only one time dimension.

In our implementation we did not provide temporal filtering. The reasons are basically the same as why we did not perform slice-timing correction. The time interval of a TR is about $2.5s$. The error from interpolation of TRs may produce much error. It always harms more than help. In today's fMRI process applications, less and less people are doing this.

If we insist on having a temporal smoothing, it can be applied in ITK::VectorImage more easily, which is the basic datatype in following GLM analysis part. See chapter 6 for more details.

4

Benchmarking of Realignment

In Insight Toolkit (ITK), as introduced before, a typical registration framework consists of six parts: *Target*, *Reference*, *Transform*, *Metric*, *Interpolator* and *Optimizer*. The basic idea is to regard a metric as cost function and apply an optimizer to find its global extreme values and update parameters in the meanwhile. Many previous works were done already on different optimizers[11][18]. Types of optimizing algorithms decides the registration quality, computation workload and robustness to noise largely.

In this chapter, we are going to test some frequently used optimizers in ITK and compare their performance, mainly in ways of computing distance to standard motion parameters (from SPM), time cost in a multi-resolution registration, and comparing their robustness to random noise. We will illustrate our experiment design in section 5.1. Then we will analyze the result of registration and discuss optimizers in the three aspects of evaluation in section 5.2. The conclusion is drawn in section 5.3.

4.1. The Experiment

In ITK, optimizers are encapsulated in ITK::ObjectToObjectOptimizer class. There are mainly 2 categories of ITK::Optimizer. See figure 4.1.

The two main categories of itk::NonLinearOptimizer are itk::MultipleValuedNonLinearOptimizer who returns multiple values, and itk::SingleValuedNonLinearOptimizer, who returns single value. However, in new versions of ITK, only single valued optimizers are available in ITKv4. In the experiment, according to single variable principle, we adopted 6 optimizers in the single-valued optimizer group. The 6 optimizers are:

- ITK::RegularStepGradientDescentOptimizerv4
- ITK::GradientDescentLineSearchOptimizerv4
- ITK::ConjugateGradientLineSearchOptimizerv4
- ITK::QuasiNewtonOptimizerv4
- ITK::AmoebaOptimizerv4
- ITK::OnePlusOneEvolutionaryOptimizerv4

We have introduce the basic mathematic theory of them in Chapter 4. Optimizers like itk::Levenberg-MarquardtOptimizer are excluded from our experiment because it returns multiple values and then are not implemented in ITKv4 registration method.

Because different optimizers are used under different conditions, for example line search optimizer can not find valid points with a mean square metric. Also because mean squared metric is very sensitive to different levels of noise. We implemented our algorithms both with mean squared metric and Mattes mutual

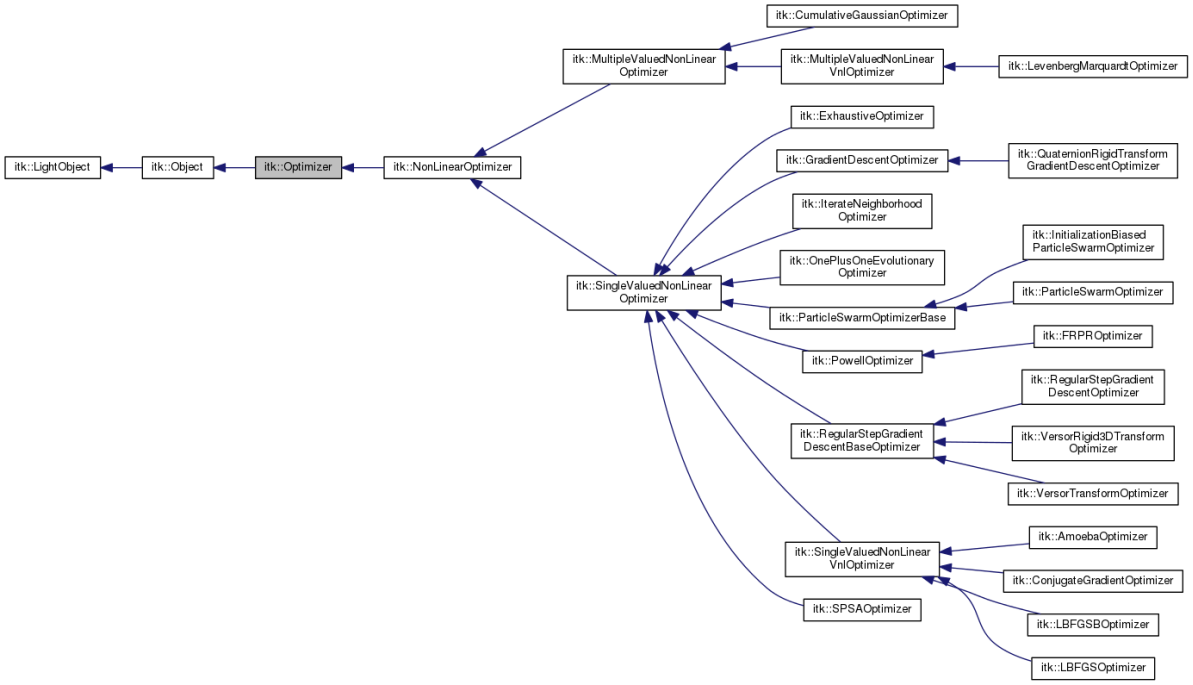

Figure 4.1: Inheritance diagram of ITK::Optimizer classes, Image from *itk.org*

information. The experiment are designed in two groups, based on single variant analysis principle. The design is shown in table 4.1. Notice that most of these optimizers find a local extrema. Therefore we applied a multi-resolution strategy of three resolution levels. Only One Plus One Evolutionary Strategy is a global optimization method. We did not apply multi-resolution to them.

Optimizers	**Metrics**
ITK::RegularStepGradientDescentOptimizerv4	ITK::MeanSquaresImageToImageMetricv4
ITK::QuasiNewtonOptimizerv4	ITK::MeanSquaresImageToImageMetricv4
ITK::AmoebaOptimizerv4	ITK::MeanSquaresImageToImageMetricv4
ITK::OnePlusOneEvolutionaryOptimizerv4	ITK::MeanSquaresImageToImageMetricv4
ITK::RegularStepGradientDescentOptimizerv4	ITK::MattesMutualInformationImageToImageMetricv4
ITK::GradientDescentLineSearchOptimizerv4	ITK::MattesMutualInformationImageToImageMetricv4
ITK::ConjugateGradientLineSearchOptimizerv4	ITK::MattesMutualInformationImageToImageMetricv4
ITK::QuasiNewtonOptimizerv4	ITK::MattesMutualInformationImageToImageMetricv4
ITK::AmoebaOptimizerv4	ITK::MattesMutualInformationImageToImageMetricv4
ITK::OnePlusOneEvolutionaryOptimizerv4	ITK::MattesMutualInformationImageToImageMetricv4

Table 4.1: Experiment Design

We apply the registration process using ITK::Euler3DTransform as our transform type, using ITK::LinearInterpolatorImageFunction as our interpolator function. As SPM does, we generate a parameter list in terms of six parameters in a matrix-offset manner, three of them are Tait-Bryan Euler angles, *pitch*, *roll* and *yaw*. The other three are translations in three dimensions. See figure 3.6.

In a real experiment, we don't have access to know how exactly the motion of a patient's head is in a scanner. As SPM is a well-developed software for fMRI image processing, we choose the result from SPM, with highest quality (7^{th} order B-Spline interpolation), as our standard parameter list. We will compute the distances between our output parameters from different optimizers and those parameters SPM provides us, to see which one performs better, or closer to SPM's result. The smaller distance to the standard, the higher quality of registration.

We define our distance between two parameter vectors in a rational way. It is intuitive to compare distances between two points, after applying spatial transfroms we got from different method respectively. The point we select should be on the cortex. We choose a point on cortex because neural activities are mainly distributed on a cortex, and it is meaningless to observe a position that is white matter inside a brain. After comparing different scanning datasets, we assume a normal human head as a sphere of radius $63mm$, centering at the origin of fMRI images. The image origins is also the rotation center of our Euler 3D transforms. In this way, we calculate the new registered point, from the 6 parameters and the initial moving point as equation 4.1. Thus, different experiments each has its own new point.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\psi & sin\psi \\ 0 & -sin\psi & cos\psi \end{bmatrix} \cdot \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix} \cdot \begin{bmatrix} cos\phi & sin\phi & 0 \\ -sin\phi & cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} \tag{4.1}$$

Then we compute the Euclidean distance between new points we calculated, defined as equation 4.2. This distance in mm shows how much the two method differ. For a long series of fMRI session images, we simply compute the average of distance over time. Generally, as the space resolution of fMRI images is about $3mm$, we assume that an error less than $3mm$ is acceptable.

$$D(SPM, ITK) = |(x'_{SPM}, y'_{SPM}, z'_{SPM}), (x', y', z')| \tag{4.2}$$

To observe the robustness of different optimizers, we generate new fMRI images by adding white noise of different variance. The noising process is done by ITK::AdditiveGaussianNoiseImageFilter. At first we set a threshold to roughly segment our head, to exclude the voxels that are other stuff like air. Then we compute average intensity of these voxels and set the standard deviation of white noise to a percentage of this average intensity value. We set the percentages in a row of power of 2, that is 1%, 2%, 4%, 8%, 16%, 32%, 64%. Therefore, we have 8 datasets (including original one) in increasing white noise in each case. See figure 4.2.

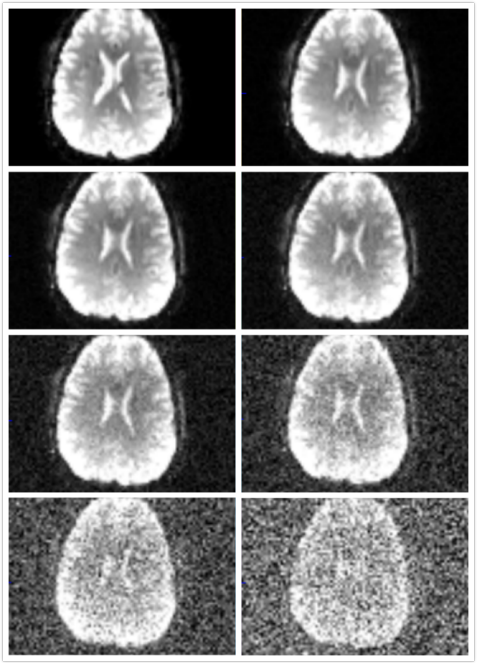

Figure 4.2: Different Level of White Noise, Generated by ITK::AdditiveGaussianNoiseImageFilter

Our experiment was on a series of 152 fMRI images. We select this dataset for its large and abrupt movement. The largest translation is about *5mm*, while the largest rotation is about *0.06rad*.See subsection 5.2.1 for more details.

The experiment was test on a laptop PC of i7-4700MQ CPU @2.40GHz, 8.00GB RAM, with Window 10 64-bits system.

4.2. Result and Discussion

As defined before, we discuss the performance of different optimizers in three aspect: Accuracy, Time and Robustness. We will illustrate them in coming three subsections.

4.2.1. Accuracy

The result from SPM on our dataset is shown in figure 4.3 and figure 4.4.

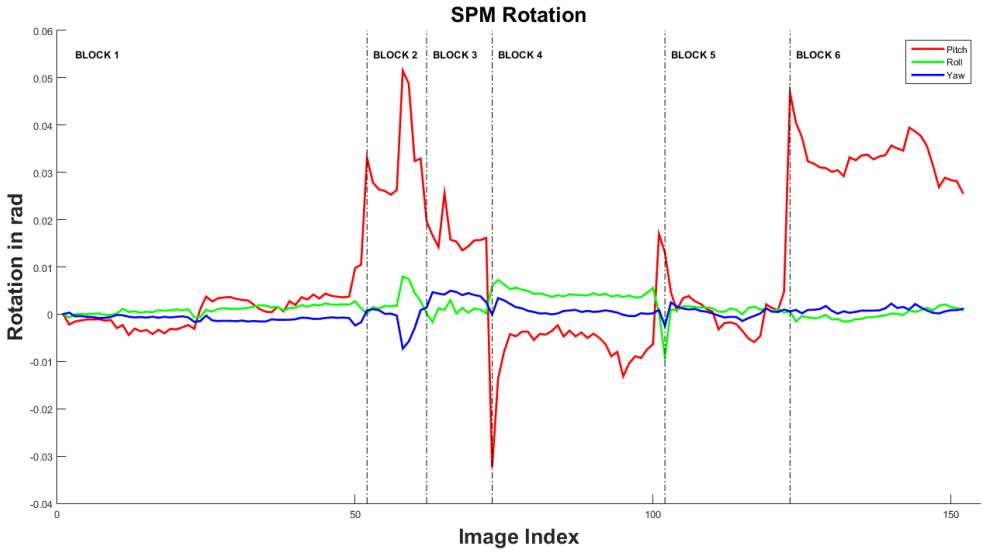

Figure 4.3: Rotation in 3 Dimensions of the Dataset from SPM, in *rad*

Figure 4.4: Translation in 3 Dimensions of the Dataset from SPM, in *mm*

As discussed before, our dataset is one with large but abrupt movement in the scanner. We selected 6 points where these abrupt movements took place. They are at 52^{nd}, 62^{nd}, 73^{rd}, 102^{nd}, 123^{rd} images. These points are useful for our discussion on performance of different optimizers. We divided the whole image series into 6 blocks. These blocks each has its own properties on 6 Euler transform parameters:

- **Block 1:** Small rotations (less than $0.01 rad$) and small translations (less than $0.5 mm$);
- **Block 2:** Large rotations (greater than $0.02 rad$) and small translations (less than $1 mm$);
- **Block 3:** Moderate rotations (about $0.02 rad$) and moderate translations (about than $2 mm$);
- **Block 4:** Small rotations (less than $0.01 rad$) and moderate translations (about than $2 mm$);
- **Block 5:** Small rotations (less than $0.01 rad$) and large translations (about $3 mm$);
- **Block 6:** Large rotations (greater than $0.02 rad$) and very large translations (about $4 mm$);

As we introduced in the experiment design, we evaluate accuracies of optimizers in term of Euclidean distance between points obtained from different methods, see equation 4.2. We set result from SPM realignment with 7-order B-spline interpolation as our standard, or *"The True Movement"*. From the dataset, we choose a point of $[0,0,63]$ as our testing landmark, which we assumed located on the cortex. For every volume images in the time series, one distance is computed. These distances are plotted along image index in figure 4.5.

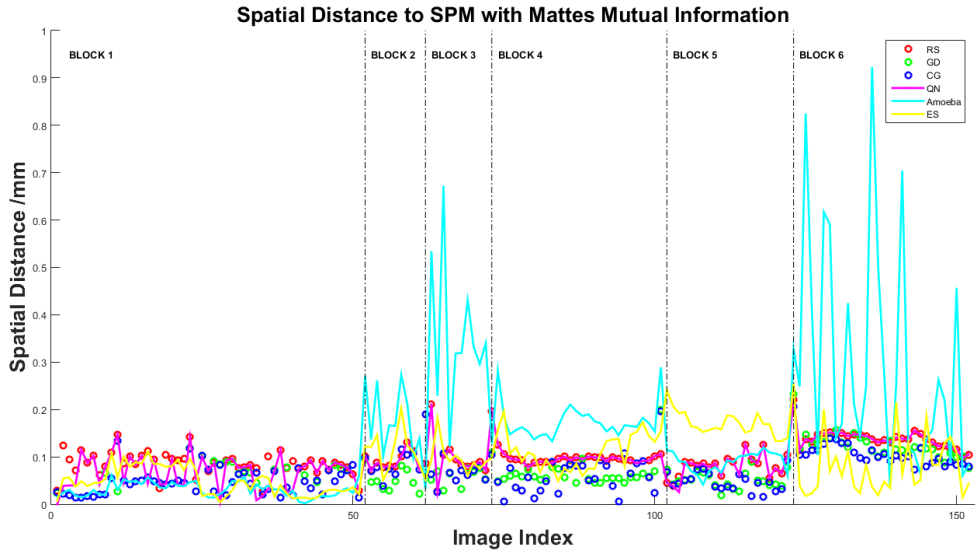

Figure 4.5: Spatial Distance of Different Optimizers with Mattes Mutual Information to SPM in mm

To compare optimizers' performance under different movement cases. We computed average distances in each blocks. The result of blocked analysis is shown in table 4.2.

Optimizers	**Block 1**	**Block 2**	**Block 3**	**Block 4**	**Block 5**	**Block 6**	**All Blocks**
Regular Step Gradient Descent	0.0858	0.0901	0.1051	0.0940	0.0888	0.1351	0.0989
Golden Section Line Search	0.0520	0.0616	0.0617	0.0650	0.0588	0.1203	0.0698
Conjugate Gradient Line Search	0.0477	0.0913	0.0683	0.0617	0.0543	0.1027	0.0661
Quasi-Newton	0.0699	0.0901	0.1051	0.0940	0.0873	0.1351	0.0932
Amoeba	0.0359	0.1584	0.3383	0.1726	0.1002	0.2963	0.1505
One Plus One Evolutionary	0.0491	0.1101	0.0984	0.1155	0.1711	0.0767	0.0915
Best Accurate Optimizer	Amoeba	GSLS	GSLS	CGLS	CGLS	(1+1)ES	CGLS
Worst Accurate Optimizer	RS	Amoeba	Amoeba	Amoeba	(1+1)ES	Amoeba	Amoeba

Table 4.2: Blocked Analysis of Different Optimizers, on Average Distance in Each Block in mm

From the blocked analysis, we can draw some conclusion about the properties of these 6 ITKv4 optimizers, working with Mattes mutual information as metric.

ITK::RegularStepGradientDescentOptimizerv4, generally performs bad in accuracy. However, it is still the most popular one in ITK applications, because of its speed, we will show it in next subsection.

ITK::GradientDescentLineSearchOptimizerv4, which applies Golden Section Line Search, perform better in cases with large rotation. Not apparent preference shown in scale of translation. Therefore, this optimzier works best in those blocks with large rotation and small translation (*Block 2* and *Block 3*).

ITK::ConjugateGradientLineSearchOptimizerv4, which applies Polak-Ribière formula, performs well all along the way, especially those blocks with large or very large translation (*Block 4*, *Block 5* and *Block 6*).

ITK::QuasiNewtonOptimizerv4, which uses BFGS method to compute inversed Hessian matrix, performs very same as ITK::RegularStepGradientDescentOptimizerv4, in some block they even have the complete same parameter lists. However, it surpass regular step method in the starting few images in a series. Normally say, Quasi-Newton optimizer is as bad as regular step optimizer in accuracy.

ITK::AmoebaOptimizerv4, which applies heuristic method to find a local optima, performs worst among these 6 optimizers. However, in a block of small movement, both in rotation and translation, it works best. We can say that this method is suitable for some small movement dataset, like in *Block 1*.

ITK::OnePlusOneEvolutionaryOptimizerv4, which applies evolutionary strategy to find a global optima, performs acceptable all along the image series. One surprising observation is that it performs best in the most large movement block (*Block 6*), which is both large in rotation and translation. We conclude that this method is suitable for large movement dataset, contrary to Amoeba method.

4.2.2. Time Cost

Apart from the accuracy, efficiency of an optimizer is another important criterion to evaluate its performance. We recorded time spent on a whole three levels multi-resolution registration (except for evolutionary optimizer) in each realignment process, on our original (without noise) dataset. All the experiments were done on the same laptop PC.

To make the results from different optimizers comparable, for 7 local optimization methods, we setup a three level multi-resolution pyramid. The shrink factors at each level are 1, 1, 2. And the smoothing factors σ at each level are 0, 1, 2. For one plus one evolutionary optimizer, which is a global optimization method, we did not apply a multi-resolution strategy. Instead, we set the maximum number of iteration to 5000, to make sure convergence. Other setups of optimizers can be found in section 5.1.

Table 4.3 shows time cost and accuracy both in different optimizer-metric pairs. Time costs are in seconds, and accuracies are in *mm*, which stands for the spatial distance between registered points, obtained from optimizer-metric pairs and SPM, as the *"True Movement"* standard.

Optimizers	**Metrics**	**Time**	**Accuracy**
Regular Step Gradient Descent	Mean Squares	7.85s	0.0671mm
Quasi-Newton	Mean Squares	3.79s	0.0667mm
Amoeba	Mean Squares	12.94s	0.0865mm
One Plus One Evolutionary	Mean Squares	67.37s	0.1054mm
Regular Step Gradient Descent	Mattes Mutual Information	14.19s	0.0989mm
Golden Section Line Search	Mattes Mutual Information	106.21s	0.0698mm
Conjugate Gradient Line Search	Mattes Mutual Information	79.29s	0.0661mm
Quasi-Newton	Mattes Mutual Information	11.25s	0.0932mm
Amoeba	Mattes Mutual Information	23.71s	0.1505mm
One Plus One Evolutionary	Mattes Mutual Information	55.85s	0.0915mm

Table 4.3: Result of Experiment test on Original (0-Noise Level) Dataset

We plotted time cost and accuracy (in spatial distance) in every case in figure 4.6. From the figure, we understand why ITK::RegularStepGradientDescentOptimizerv4 and ITK::MeanSquareImageToImageMetricv4 pair is the most popular optimizer-metric pair in ITK applications. It works pretty good in accuracy and it is very efficient in time. ITK::QuasiNewtonOptimizerv4 has a better performance than regular step one. It is faster and a little more accurate than regular step optimizer.

However, in very noisy images, or multi-modality image registration. Means square metric is not sufficient in accuracy any more. When we adopted ITK::MattesMutualInformationImageToImageMetricv4 as our metric, the choice is harder to make. As normal, there is a trade-off between quality and speed. Line search methods will take much longer time, but give a better registration.

In our case, the spatial distance is around $0.1mm$, which is much smaller than a typical spatial resolution of fMRI images. Therefore, all the registrations are acceptable. An optimizer that takes less time is preferred.

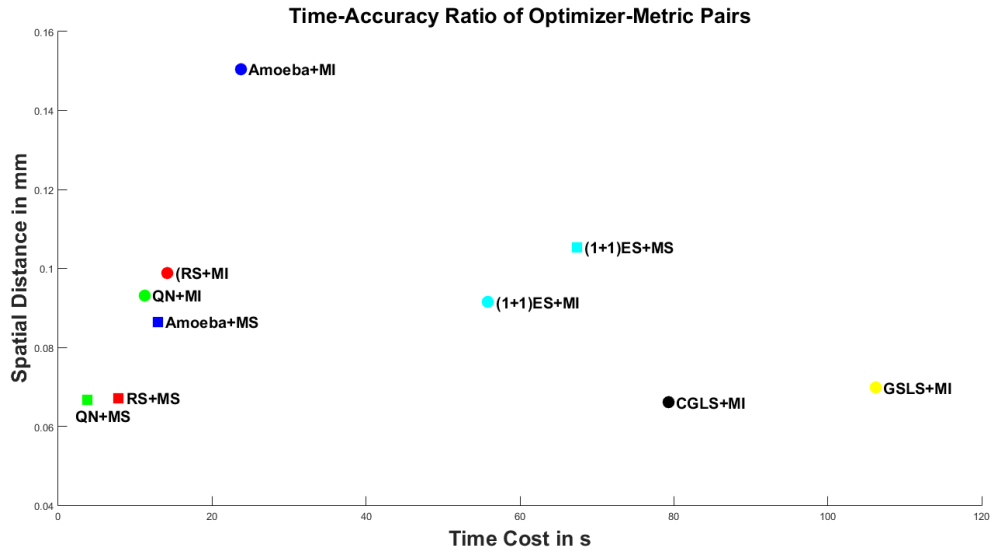

Figure 4.6: Spatial Distance and Time Cost for a Registration of Different Optimizers

4.2.3. Robustness to Noise

The third important measure of performance is robustness to noise of optimizers. As we introduced before, the experiments are in 8 groups, each group using a dataset generated by adding different levels of white noise to the original one.

We have 8 dataset as 0% (original), 1%, 2%, 4%, 8%, 16%, 32%, 64%. All the ten experiments described in table 4.1 are done. Also, we used SPM as our 11^{th} experiment. Therefore we have 88 tests in total. We computed distance to *the true movement* in each case. As we said, *the true movement* is obtain from SPM with original dataset. The results are shown in table 4.4.

We plot the results in a figure to show their robustness to noise in figure 4.7 (left). We selected all the experiments with derivative-based optimizers in 4.7 (right). From the figures we can find following:

- When same optimizer is applied, Mattes mutual information metric shows better robustness to noise, compare to mean square metric.
- Amoeba optimizer and one plus one evolutionary optimizer have a bad robustness to noise. especially Amoeba methods. That is derivative-based methods are more robust than heuristic methods.

Experiments	0%	1%	2%	4%	8%	16%	32%	64%
RS+MS	0.0671	0.0663	0.0707	0.0641	0.0652	0.0749	0.0777	0.1832
QN+MS	0.0667	0.0663	0.0703	0.0641	0.0647	0.0749	0.0768	0.1832
Amoeba+MS	0.0865	0.0897	0.0925	0.1021	0.1923	0.3281	1.4617	4.6412
(1+1)ES+MS	0.1054	0.1023	0.1020	0.0983	0.0979	0.2193	1.1270	1.5005
RS+MI	0.0989	0.1009	0.1211	0.1077	0.1015	0.0989	0.0959	0.1469
GSLS+MI	0.0698	0.0636	0.0581	0.0636	0.0799	0.0911	0.1942	0.3364
CGLS+MI	0.0661	0.0625	0.0599	0.0701	0.0955	0.1101	0.2256	0.8279
QN+MI	0.0932	0.0967	0.1122	0.1062	0.1007	0.0987	0.0958	0.1602
Amoeba+MI	0.1505	0.1091	0.1086	0.1704	0.2897	0.7886	3.0343	4.1101
(1+1)ES+MI	0.0915	0.0972	0.0927	0.1015	0.1044	0.1181	0.4005	1.5045
SPM	N.A.	0.0487	0.0497	0.0488	0.0473	0.0660	0.0877	0.1964

Table 4.4: Result of Robustness Experiment test on Small Movement Dataset, all results in mm, which stands for the distance between registered points from tested optimizer and SPM.

- Generally, mean square metric works better than Mattes mutual information in accuracy. That is images in our dataset the are from a same scanning session, and the scanning conditions are not varies much. Mattes mutual information uses Parzen estimation to compute probability distributions, that may lead to more errors. However, in a more general case, Mattes mutual information metric
- In all the derivative-based methods we tested, regular step gradient descent and quasi-Newton method works much better in robustness. Although the accuracy of them is worse than line search methods in low noise datasets, the accuracy became comparable even less than line search methods.
- SPM has its high-performance algorithms for realignment, with a good robustness to noise. However, our experiments showed that gradient descent and quasi-Newton methods have a better robustness. In high-noise dataset, the accuracies of them are even better than SPM.

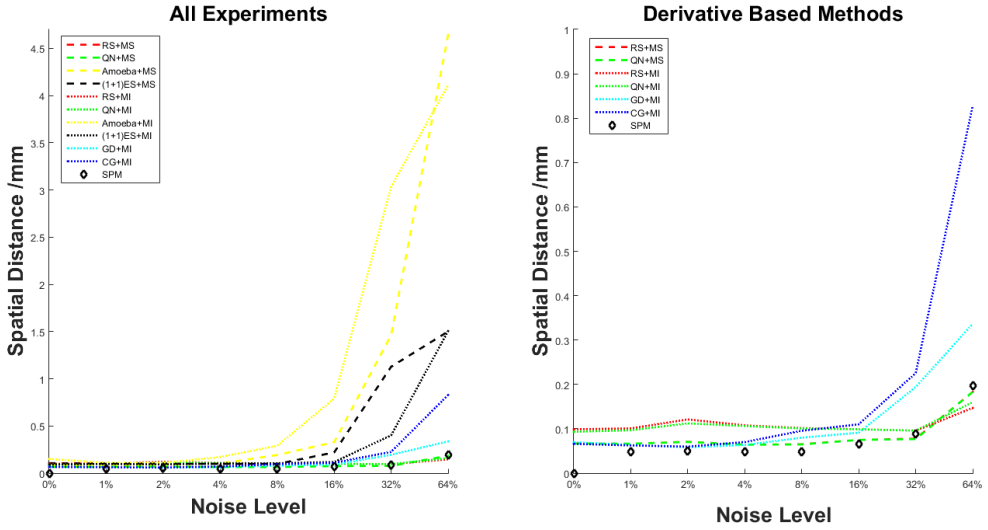

Figure 4.7: Spatial Distance and Time Cost for a Registration of Different OptimizersRobustness of Optimizers to Different Level Noise

After compared these experiments, we obtained some instructions on how to select a proper (accurate and efficient) optimizer-metric pair when the dataset is ruined by noise.

When we are facing low-noise dataset, heuristic methods and derivative-based methods both works fine. Line search methods have the best accuracies.

When we are facing high-noise dataset, we prefer derivative-based method. When the noise level is growing up to 8% to 16%, Amoeba optimizer and evolutionary optimizer start to fail. When the noise level is higher

than somewhere near 16%, line search methods starts to have a larger error than regular step gradient descent optimizer do.

4.3. Conclusion

From the experiments, we have some conclusion on in some case which optimizer is preferred to be the best choice in realignment:

In low-noise dataset, line search optimizers generally have the best accuracies, but cost too much time. Amoeba and evolutionary optimizers are used in some extreme cases. Amoeba optimizer performs best in very small movement dataset, while evolutionary optimizer performs best in very large movement dataset. In normal cases, regular step gradient descent or quasi-Newton optimizer would be a better choice. Quasi-Newton optimizer is better than regular step gradient descent optimizer in every sense.

In high-noise dataset, that is the noise level is higher than 16% of the average intensity of the subject, line search methods and heuristic methods both fail.

To conclude, if we don't have any information or clues of a dataset before realignment, quasi-Newton optimizer is always a good choice, in accuracy, in speed and also in robustness to noise.

5

Basic Statistic Analysis

After pre-processing, we obtained fMRI images that are re-sliced into a same spatial grid. We then move to some basic statistical analysis of fMRI data.

In this chapter, we first introduce the most widely used *General Linear Model* method. In GLM, we estimate the fitting parameters to minimize squared error. We will also introduce hypothesis testing methods, mainly the statistical parameteric mapping of *student's t_distribution*. We can estimate a t-map of a 3-D brain image, to see the significance of activation related to task or events. Notice that all discussions and implementations in this chapter are in sense of individual voxels. When we like to analyze the brain activities dependently, that is a multiple comparison problem, we will discuss them in chapter 7. All methods above iterates an image and analyze one voxel at a time. This is called *univariate* methods. If we analyze more than one voxel simultaneously, that is *multivariate* method, which is beyond this project.

5.1. General Linear Model

General Linear Model (GLM) is a generalization of linear regression method. We want to find how much is a BOLD signal significantly related to the task model. The method helps find Brain regions show strong task-related activity.

Two popular methods that models BOLD signal are introduced in this chapter: correlation-based methods and finite BOLD response. These two model both lead to a linear regression problem. We will first introduce the two modeling methods, and then discuss how to estimate parameters in the regression problem.

5.1.1. Finite BOLD Response

Finite BOLD Response (FBR) is similar as FIR in signal theory. "Finite" means that FBR is non-zero only at limited number of TRs. The influence of a BOLD response is constrained in time[1]. Suppose an event or task influences the BOLD signal in following t seconds, that is $\frac{t}{TR}$ TRs in an fMRI image series. On the contrary, observed BOLD signal at one TR has contribution from several events or tasks occur at previous TRs. Suppose event E_1 has an FBR last 5 TRs. The values are $\beta_{1i}, i \in [1,5]$. And event E_2 has an FBR last 3 TRs. The values are $\beta_{2j}, j \in [1,3]$. We can sum up contributions of each events to BOLD signal at each TR. See a table 5.1 below:

In the table we added two rows: a constant BOLD and a linear drift BOLD. When we apply a constant BOLD, the β will become near zero if the voxel is not event-related. We added a drift vector to compensate the non-stationarity of magnetic field in a scanning session. The variance becomes larger along the scanning session. The design can be written in a linear matrix equation 5.1. By solving this GLM equation at each voxel, we can get these β parameters that shows how significant the voxel is activeted by the events or tasks.

We can assume a large upper bound for an FBR, to make sure the response is not truncated. This generates more parameters, which will reduce the degree of freedom in following t-map hypothesis testing. See

TR:	TR_1	TR_2	TR_3	TR_4	TR_5	TR_6	TR_7	TR_8
Events:	E_1	E_2			E_2		E_1	
BOLD of $1^{st}E_1$	β_{11}	β_{12}	β_{13}	β_{14}	β_{15}			
BOLD of $1^{st}E_2$		β_{21}	β_{22}	β_{23}				
BOLD of $2^{nd}E_2$					β_{21}	β_{22}	β_{23}	
BOLD of $2^{nd}E_1$							β_{11}	β_{12}
Constant BOLD	B_0	B_0	B_0	B_0	B_0	B_0	B_0	B_0
Drift BOLD	Δ	2Δ	3Δ	4Δ	5Δ	6Δ	7Δ	8Δ

Table 5.1: An Example of Event-Related Design Experiments and its BOLD Constitution

6.2 for more details. It will also increase the computation complexity. However, it is even worse if the upper bound of an FBR is less than what it should be. A small upper bound truncated the response. This lost response will be absorbed by following events or error terms. If we are using FBR method, we should always keep conservative to avoid underestimation of FBR duration.

$$\begin{bmatrix} B(TR_1) \\ B(TR_2) \\ B(TR_3) \\ B(TR_4) \\ B(TR_5) \\ B(TR_6) \\ B(TR_7) \\ B(TR_8) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 4 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 6 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 7 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 8 \end{bmatrix} \cdot \begin{bmatrix} \beta_{11} \\ \beta_{12} \\ \beta_{13} \\ \beta_{14} \\ \beta_{15} \\ \beta_{21} \\ \beta_{22} \\ \beta_{23} \\ B_0 \\ \Delta \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \\ \varepsilon_6 \\ \varepsilon_7 \\ \varepsilon_8 \end{bmatrix} \tag{5.1}$$

In our implementation, we did not use FBR method. We applied correlation method instead.

5.1.2. Correlation Method

Apart from FBR, another frequently used modeling method is correlation method. Correlation-based methods was invented earlier than FBR in 1990's. The idea of correlation method is to combine all events of same stimulation to a neural activation function. Then convolve this neural activation function with a predefined *Hemodynamic Response Function (hrf)*. We sample the convolution at each TR as our predicted BOLD value. See 5.2, neural activation function $N(t)$ is the boxcar shape neural activation function, and $h(t)$ is a predefined hrf.

$$x(t) = N(t) * h(t) = \int_0^t N(\tau)h(t-\tau)d\tau \tag{5.2}$$

The predefined hrf $h(t)$, there are many options. According to the real response properties of a BOLD signal, a γ function by *Boynton* et al (1996) is sometimes preferred, see figure 5.1 left. *Friston* et al. (1998) assumes hrf to be a linear combination of some fixed basis functions (for example γ funxtion). In SPM, a function spm_hrf let users to define their own hrf by setting some parameters like onset, delay of response etc.

The neural activation functions $N(t)$ are always chosen as a boxcar function. Because the time points of TRs are discrete, we define the activation function equals to 1 from a TR where an event occur to the beginning of next TR. If at a consecutive number of TRs occurs a same event, the activation function becomes 1 during these TRs. That is what called a *block design*, which is actually a special case of event-related design. Our experiments were done under a block design. The experiment was designed to find the brain region that is related to stimulations to left and right eyes. In the experiment, every stimulation last for 20 seconds. The stimulations to left eye onsets on $[0, 40, 80, 120, 160, 200, 240, 280, 320]$, while the rest time was occupied by stimulations to right eye. The stimulation pairs are repeated for nine times, thus the total time of the session is 360 seconds. We took 195 EPI images during this 360 seconds, the TR was 1.89 seconds. Figure 5.1 mid shows the neural activation function in our testing dataset. Figure 5.1 right shows the convolution of neural

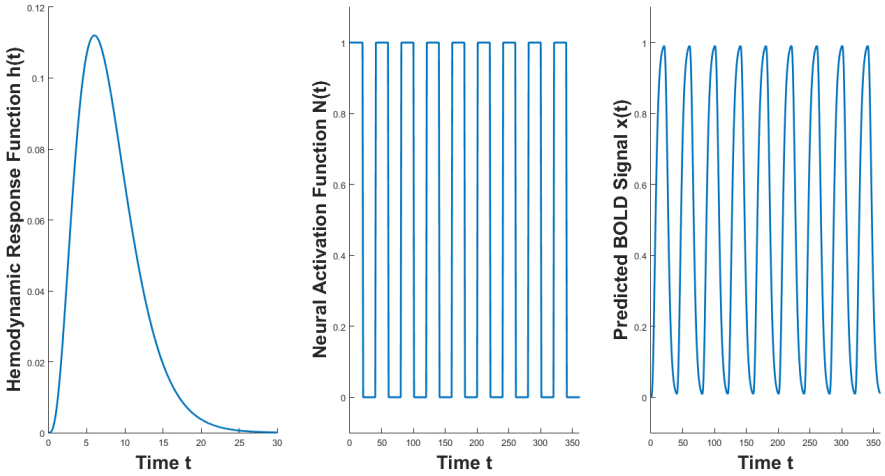

Figure 5.1: Left is a Typical Hemodynamic Response Function: A Gamma Function by Boynton 1996. Mid is Neural Activation Function in Our Experiments Dataset. Right is Convolution of $N(t)$ and $h(t)$, the Predicted BOLD Signal

activation function and hrf, which is our predicted BOLD signal.

An observed BOLD signal is also a combination of different models. We write it in a form of matrix equation as 5.3. We see that the number of parameters reduces compared to FBR methods, and the assumptions in correlation methods is also more strict than in FBR methods. We define the model matrix before we estimate a most fitting β vector, which minimize ε terms. We will see in next section.

$$\begin{bmatrix} B(TR_1) \\ B(TR_2) \\ B(TR_3) \\ B(TR_4) \\ B(TR_5) \\ B(TR_6) \\ B(TR_7) \\ B(TR_8) \end{bmatrix} = \begin{bmatrix} x_1(TR_1) & x_2(TR_1) & 1 & 1 \\ x_1(TR_2) & x_2(TR_2) & 1 & 2 \\ x_1(TR_3) & x_2(TR_3) & 1 & 3 \\ x_1(TR_4) & x_2(TR_4) & 1 & 4 \\ x_1(TR_5) & x_2(TR_5) & 1 & 5 \\ x_1(TR_6) & x_2(TR_6) & 1 & 6 \\ x_1(TR_7) & x_2(TR_7) & 1 & 7 \\ x_1(TR_8) & x_2(TR_8) & 1 & 8 \end{bmatrix} \cdot \begin{bmatrix} \beta_1 \\ \beta_2 \\ B_0 \\ \Delta \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \\ \varepsilon_6 \\ \varepsilon_7 \\ \varepsilon_8 \end{bmatrix} \tag{5.3}$$

5.1.3. Parameter Estimation

From the two previous sections, both in FBR method and correlation method cases, we need to estimate a best fitting β vector to minimize the error terms ε.

We define the observed BOLD signal, as Y. The model matrix as X. Y is a vector of intensity scalars in fMRI images. Each column in X is a predicted BOLD response of a certain task. Thus the matrix equation can be written as equation 5.4.

$$\begin{cases} Y = X\beta + \varepsilon \\ \beta = \arg\min_{\beta}[(Y - X\hat{\beta})'(Y - X\hat{\beta})] \end{cases} \tag{5.4}$$

We want to minimize the error term ε. We compute the *Sum of Squared Error (SSE)*. According to *Gauss-Markov Theorem*, we can estimate β vector value as equation 5.5.

$$\hat{\beta} = (X'X)^{-1}X'Y \tag{5.5}$$

In our implementation, we adopted ITK::VectorImage to store all the 195 EPI images in one ITK object. The ITK::VectorImage shares spatial information with all the re-sliced fMRI images. The only difference is that the value at each voxel is a vector of length 195 instead of a single intensity scalar. We used

ITK::ComposeImageFilter to push intensities into the vector along TRs.

As for the model matrix X, we used eight models. The first model is the neural activation function $N(t)$ (figure 5.1 mid) sampled at each TR, as an approximation of real predicted BOLD response $x(t)$ (figure 5.1 right). The second and third model are a constant model and a drift model, which are often used. The last 5 models are generated from *Discrete Cosine Transform (DCT) basis set*, as a temporal high-pass filter. We applied DCT basis in order of 5.

$$\begin{bmatrix} I(TR_1) \\ I(TR_2) \\ I(TR_3) \\ \vdots \\ I(TR_{195}) \end{bmatrix} = \begin{bmatrix} N(TR_1) & 1 & 1 & cos(\pi/195) & cos(2\pi/195) & \cdots & cos(5\pi/195) \\ N(TR_2) & 1 & 2 & cos(2\pi 195) & cos(4\pi/195) & \cdots & cos(10\pi/195) \\ N(TR_3) & 1 & 3 & cos(3\pi/195) & cos(6\pi/195) & \cdots & cos(15\pi/195) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ N(TR_{195}) & 1 & 195 & cos(\pi) & cos(2\pi) & \cdots & cos(5\pi) \end{bmatrix} \cdot \begin{bmatrix} \beta \\ B_0 \\ \Delta \\ \beta^{(1)} \\ \beta^{(2)} \\ \beta^{(3)} \\ \beta^{(4)} \\ \beta^{(5)} \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \vdots \\ \varepsilon_{195} \end{bmatrix} \tag{5.6}$$

The DCT basis function of lowest frequency takes time of the whole session (360 seconds) as half of its period. And the upper bound of frequency is normally decided by the double time between same onset in experiment block designs. In our case, the stimulations are repeated every 40 seconds. Thus the shortest period should be larger than 80 seconds. See figure 5.2.

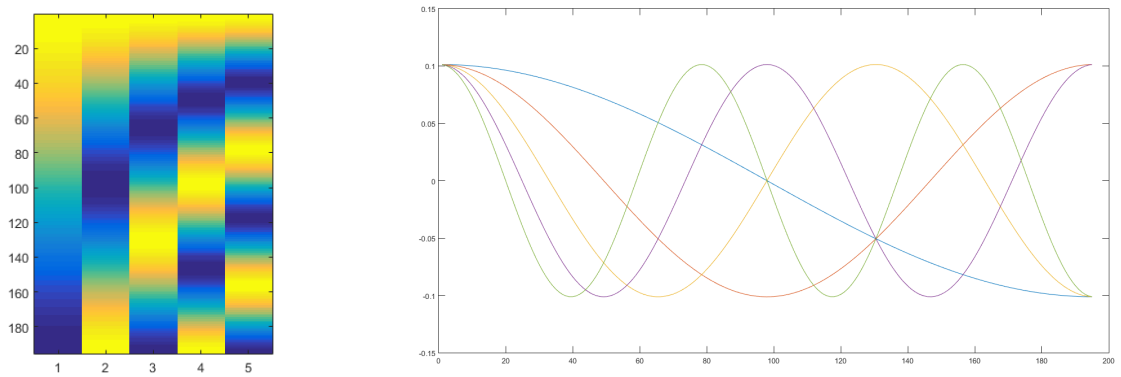

Figure 5.2: Discrete Cosine Transform Basis Functions, visualized with SPM

We iterated all the voxels in the ITK::VectorImage (151552 voxels in our experiments) and computed β vectors for each of them. Thus we generated 8 β-maps. The value of B_0-map is around the average intensity of this voxel along TRs. We call it baseline activation. Δ and $\beta^{(i)}$ absorbed unexpected noises from β-map. The value of β-map shows if the voxel is significantly related to the eye stimulation. A β near zero shows the voxel is less related to the eye stimulation. A β that is apparently larger than zero implies that the voxel is activated when left eye stimulation is conducted. On the contrary, A β that is apparently smaller than zero implies that the voxel is activated when right eye stimulation is conducted.

5.2. Hypothesis Testing

To see the significance of the relation between activation and task, we appealed to statistical hypothesis testing. In fMRI cases, the alternative hypotheses are often chosen as *"a brain region is related to a task or stimulation"*, on the opposite, a null hypothesis is chosen as *"a brain region is uncorrelated with a task or stimulation"*. If a region that is actually unrelated to a task, but we regarded it as related, we are making *Type I Error (False Positive)*. On the contrary, if a region that is actually related to a task, but we regarded it as unsignificant, we are making *Type II Error (False Negative)*. In our experiment, we setup our hypothesis like 5.7. The motation of β here is a value of first model ($N(t)$), not a β vector.

$$\begin{cases} H_0 : \beta \approx 0 \\ H_1 : \beta > 0 \end{cases} \tag{5.7}$$

In more complex conditions, for example more than one task or stimulation, the hypotheses also becomes more complicated. A simple saying is not sufficient. Here we introduce a contrast vector over models. The contrast vector has a length of number of models. Although we only have on task, we generalized our hypotheses to 5.8, where $[1,0,0,0,0,0,0,0]$ is the contrast vector, denoted as c.

$$\begin{cases} H_0 : c'\beta \approx 0 \\ H_1 : c'\beta > 0 \end{cases} \tag{5.8}$$

5.2.1. Student's t_Test

Before we move on, we would introduce *Student's t_Distribution* a bit. Student's t_Distribution, invented by *William Sealy Gosset*, is a variant of normal distribution. It is applied when small population of samples are available. While normal distribution describe the full population, t_distribution describes a set of samples from full population. When the amount of samples gets larger, the shape of t_distribution gets more similar to normal distribution. The measure of amount of samples is notated as concept of *Degree of Freedom*.

Back to our experiment, we compute the distribution of $c'\beta$, under the null hypothesis H_0. However, $c'\beta = c'(X'X)^{-1}X'B$, where c and X are all constants. $c'\beta$ is actually a linear transform of $B = X\beta + \varepsilon$. Therefore, the expectation of $c'\beta$ is zero under null hypothesis. We then compute the variance of $c'\beta$. See 5.10, here DF_E is degree of freedom, which is defined as number of images minus number of models.

$$\begin{cases} Var[c'\beta] = c'(X'X)^{-1}cVar[X\beta+\varepsilon] \\ Var[X\beta+\varepsilon] = \dfrac{1}{DF_E}(B - X\hat{\beta})'(B - X\hat{\beta}) \end{cases} \tag{5.9}$$

Therefore, we can compute a t-value at each voxel, from its β vector and our predefined contrast vector c.

$$t = \frac{c'\beta}{\sqrt{\frac{1}{DF_E}(B - X\hat{\beta})'(B - X\hat{\beta})c'(X'X)^{-1}c}} \tag{5.10}$$

It is notated as t-value because it satisfies a t_distribution whose degree of freedom is number of TRs minus number of models. In our experiment, the degree of freedom is $195 - 8 = 187$.

5.2.2. Statistical Parametric Mapping

Up to now, we have computed a t-value for each voxel in a 3-D image. We map them to generate a t-map. We here simply apply 3 as an rough thresholding and see the brain region that is significant related to the sys stimulation tasks. The region that is identified as unrelated to task has a value thresholded to 0. Regions with a negative value less than -3 is identified as significantly related to left eye stimulation, which is in red, while regions with a positive value larger than 3 is identified as significantly related to right eye stimulation, which is in blue. We visualize them together with the co-registered anatomical image we gained from co-registration in figure 5.3.

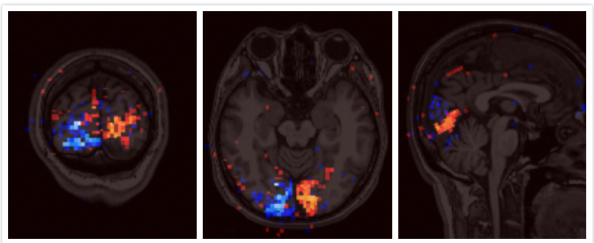

Figure 5.3: Localization of Activated Region with a Higher T-Value than 3, from left to right coronal, axial and sagittal.

We can see from the localization that the region related to eye stimulation is at rear part of a brain, which is identical to what neuroanatomy tells us. Also, the activation region related to left-eye stimulation located at the right half of the brain, while the activation region related to right-eye stimulation located at left half of the brain, as predicted.

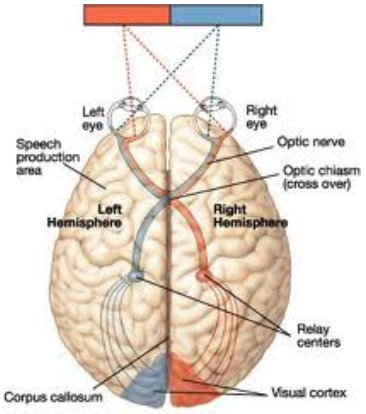

Figure 5.4: Illustration of Vision Cortex

We also tested on noisy datasets. The noisy datasets are generated by ITK::AdditiveGaussianNoiseImage-Filter as well. The noise level are also the same as we did in realignment benchmarking. From figure 5.5 we may notice that the activation region is getting smaller when a dataset becomes more noisy. When the noise level is higher than 16%, we basically can not see a significant region that is related to the task.

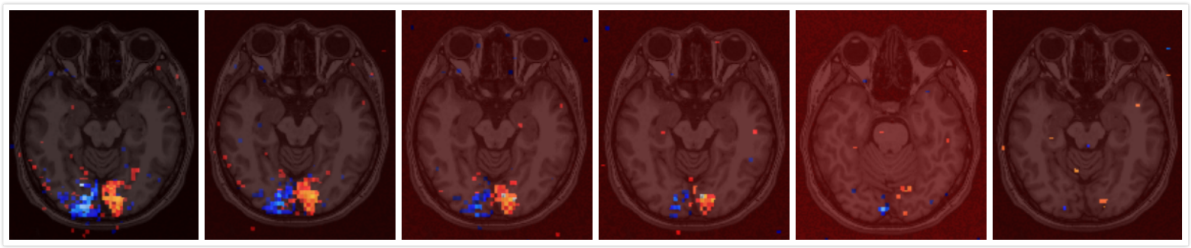

Figure 5.5: t-Maps of Datasets with Different Noise Levels, Thresholded at 3, from left to right the noise levels are 0%, 1%, 2%, 4%, 8%, 16%

However, simply apply a plain digit as threshold is not sufficiently convincing in fMRI researches. There are many more scientific methods to select a proper threshold for a t-map to identify significance. We will discuss them in next chapter.

6

Multiple Comparisons

In chapter 6, we are already able to construct a t-map of a brain, from an fMRI series. What we did is simply choose a plain value, 3, as a threshold. In statistics, we often compute the threshold by limiting a false-positive rate. This chapter will discuss how do we use this t-map to find a robust threshold, revealing if a voxel is related to a task or stimulation.

6.1. Single Voxel Cases

In our experiments we have 151552 voxels in a t-map, thus we have 151552 same t-distributions as well. These distributions, whose mean is zero, and shape decided by the degree of freedom, was constructed under the null hypothesis that one particular voxel is not responsive to the task or stimulation. The observed t-value is a random variable. When an absolute t-value is higher than threshold somewhere, we reject null hypothesis at this particular voxel, then say this voxel as responsive.

However, although the simple t-statistics works for single voxel, it fails when there is a large amount of voxels. Suppose we have a t-map generated from a very noisy fMRI series, like 64% noise in our experiments. Not point is showing significance. However, if we set a threshold based on false-positive rate, for example $\alpha = 0.05$, there will be a smaller threshold and over 7500 voxels are randomly identified as significantly related to task, which is actually not. *Bennett* did a famous dead salmon experiment to show this ridiculous results: a dead salmon shows activation to some stimulations. Separately doing statistical analysis for each voxel may lead to a *Multiple Comparisons Problem*. To prevent from this problem, a new criterion was proposed, *experiment-wise false-positive rate*, α_E, instead of a simple false-positive rate α. In next three sections, we will discuss some frequently used methods in multiple comparison correction.

6.2. Bonferroni Correction

Suppose all the voxels are independent mutually, *Bonferroni* correction was proposed. See equation 6.1, N is the amount of separate and independent tests. In our case, N is 151552, the number of voxels.

$$\alpha_t = 1 - (1 - \alpha_E)^{1/N} \approx \alpha_E / N \tag{6.1}$$

If we choose experiment-wise false-positive rate $\alpha_E = 0.05$, the real threshold on t-value α_t is approximately 3.3×10^{-7}, whose corresponding t-value is 5.149, which is much larger than 1.653 in single voxel case.

Bonferroni correction has an assumption that all voxels are independent mutually. However, this assumption is not satisfied in fMRI images. fMRI images are spatially correlated. It is intuitive that neighboring voxels of an activated voxel is more potential to be activated. Assuming independent makes Bonferroni method too strict and conservative when choosing a threshold on t-value.

6.3. False Discovery Rate

Before, we threshold t-values by satisfying an experiment-wise false-positive rate α_E, which is the probability that at least one false-positive occurs in the whole volume. However, *Benjamini* and *Hochberg* proposed that another proportion is more important[3]. It is the false-positive in all significant voxels, which is called *FDR (False Discorvery Rate)*. See table 6.1. We define the FDR as equation 6.2.

	Accept	Reject	Total
Non-Activated	U	V	N_0
Activated	T	S	N_1
Sum	W	R	N

Table 6.1: Outcomes of N Tests of the Null Hypothesis of Non-Activated

$$FDR = E[\frac{V}{R}] \tag{6.2}$$

Here is how *Benjamini* and *Hochberg* algorithm works. At first, convert all t-values in to its probabilities p-values. Each voxel has one p_i. Then we order these p_i increasingly in a vector $(P[1], P[2], P[3], \ldots, P[N])$. At last, find the largest k that satisfying 6.3. Here q is our threshold. Voxels whose index is less than q are identified as significant, where we reject our null hypothesis. Voxels whose index is larger than q are regarded as non-activated.

$$P[k] < qk/N \tag{6.3}$$

Our project implemented multiple comparison correction using FDR. We applied *Boost* C++ libraries to support conversion between t-values and probabilities.

6.4. The Whole Pipeline

Up to now the whole pipeline of fMRI pre-processing and basic statistical analysis is constructed. We did an experiment to test the whole pipeline, from original data directly from scanner, to our activation map. The experiments are also designed in different levels of noise, from 1% to 16%.

For the realignment part, we applied ITK::QuasiNewtonOptimizerv4 for its overall best performance. We re-sliced the fMRI images and computed β-maps and t-maps, according to the onsets and TRs of the eye stimulation experiment, see chapter 6. The t-maps we generated from FIONA is shown in figure 6.1.

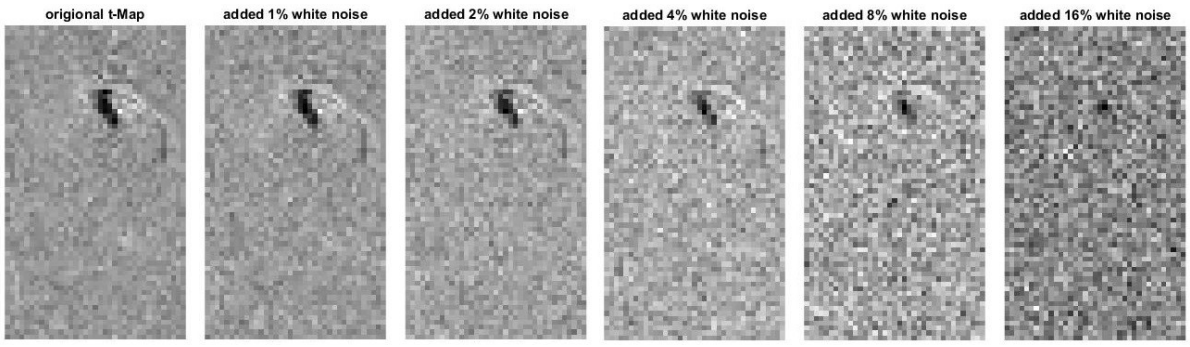

Figure 6.1: Noisy t-Maps Used to Test FDR Implementation, the t-maps are generated by FIONA.

Then we applied our FDR implementation to these t-maps and computed threshold of t-values in each case. We may see them in table 6.2.

t-Map	t Threshold	p-Value
Original fMRI	3.63018	0.000182752
1% Noise fMRI	3.73879	0.000122832
2% Noise fMRI	3.91150	0.000064130
4% Noise fMRI	4.19512	0.000021040
8% Noise fMRI	4.69725	0.000002547
16% Noise fMRI	No Significant Activation	

Table 6.2: Outcomes of N Tests of the Null Hypothesis of Non-Activated

From the results,we can find that the software FIONA works against a noise level somewhere between 8% and 16%. From 16% on, the original fMRI data are seriously corrupted and the FIONA stops working. Generally the intensity noise in original fMRI data is less than 5%. Thus, the robustness against noise of the whole pipeline is acceptable.

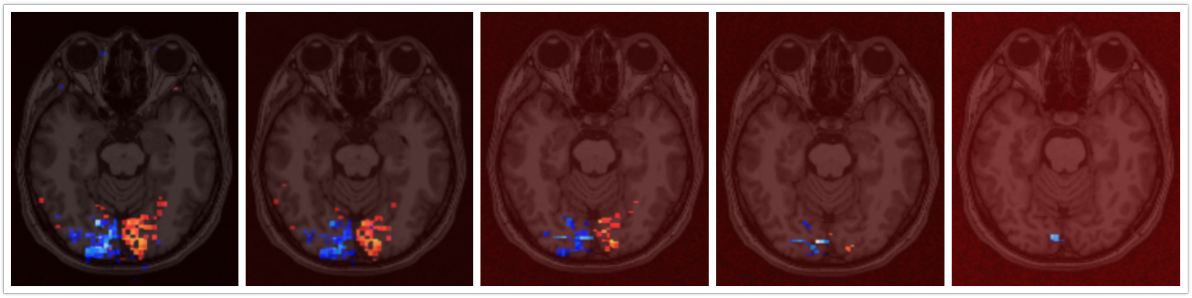

Figure 6.2: t-Maps of Datasets with Different Noise Levels, Thresholded at their threshold computed from FDR algorithm, from left to right the noise levels are 0%, 1%, 2%, 4%, 8%.

For the 16% dataset, after FDR not even one voxel is assumed as significantly responsive, to control the false discovery rate. Although the activated voxels are less, the responsive regions are smaller, the possibilities of a voxels that is out of the responsive area to be regarded as significant is much smaller. We can see from these five figures that almost all activated voxels are located in the region, which is the actual ones responsive to eye stimulation. Remember figure 5.5, the activation regions are larger, but there are much more voxels outside of the region are assumed to be activated.

7

Summary

Current version of FIONA is ready for clinical uses in conjunction with *Neural Navigator*, even though there are many things could be added on in the future.

We designed FIONA with preprocessing functionalities including realignment, co-registration and spatial smoothing. We found the overall best ITKv4 optimzier, the ITK::QuasiNewtonOptimizerv4, via a benchmarking, and implemented in FIONA. Also, we applied the most proper optimizers in our multi-stage co-registration implementation. We applied spatial smoothing when we re-slice realigned images. In the future, we may incorporate time-slicing correction and normalization into coming versions of FIONA.

We also designed a basic statistical analysis implementation, General Linear Model, and multiple comparison correction, False Discovery Rate Correction. FIONA is able to generate statistical parametric maps, if models and contrast are provided. The map directly shows the brain region of the patient, that is responsive to tasks or stimulations in the experiment. In the future, we may provide other statistical analysis methods, for example *DCM* or other multiple comparison correction approach, like *Gaussian Random Filed Correction* from FIONA.

We designed and developed an UI for FIONA. Users are able to interact with FIONA in an easy way. FIONA can visualize the statistical parametric maps together with high resolution anatomical MRI scans. Also, FIONA is able to color map the statistical parametric maps to a reconstructed brain cortex.

A

NIfTI Files

The data we test for our implementation were in format of NIfTI. NIfTI is short for *Neuroimaging Informatics Technology Initiative*. NIfTI-1 is a data format storing neuroimaging data (see http://nifti.nimh.nih.gov/). The suffix of NIfTI files .hdr/.img or .nii. In our implementation, we used NIfTI-1 data format as default. Other format like DICOM can be easily converted to NIfTI-1 file.

The NIfTI-1 format comes from previous ANALYZE 7.5 data format, who lacks adequate information about orientation is space. As inherited from ANALYZE 7.5 format, NIfTI also uses .hdr file to store metadata in a header, uses .img to store the image data, together with a extension of 4 bytes. Surely these two files should be used in pairs. A more convenient way is to store the image and header in a single .nii file.

A NIfTI header is import to store spatial information of an image. A NIfTI header has 348 bytes. Between a header and image data, there is an extension, which is always 4 bytes. The fields of 348 bytes header is shown in table A.1. The detail info of each field can be found at *nifti.nimh.nih.gov/nifti-1*. Here, we only discuss some fields accessed in our implementation.

The spatial orientation information are stored in the header from 252 bytes to 327 bytes. There are two representation of the spatial information: a quaternion expression and a rotation matrix expression. qform_code and sform_code specify what is the expression used for respectively. Normally is the spatial information of an image stored is directly from the scanner, the code is 1. If the expression stored is a co-registered one, the code is 2. Conventionally, we store the original orientation from scanner in the quaternion, and store the registered orientation in rotation matrix, after realignment. We will introduce how we re-write a NIfTI-1 header in 4.2.5.

TYPE	NAME	OFFSET	SIZE	DESCRIPTION
int	sizeof_hdr	0B	4B	Size of the header. Must be 348 (bytes).
char	data_type[10]	4B	10B	Not used; compatibility with analyze.
char	db_name[18]	14B	18B	Not used; compatibility with analyze.
int	extents	32B	4B	Not used; compatibility with analyze.
short	session_error	36B	2B	Not used; compatibility with analyze.
char	regular	38B	1B	Not used; compatibility with analyze.
char	dim_info	39B	1B	Encoding directions (phase, frequency, slice).
short	dim[8]	40B	16B	Data array dimensions.
float	intent_p1	56B	4B	1st intent parameter.
float	intent_p2	60B	4B	2nd intent parameter.
float	intent_p3	64B	4B	3rd intent parameter.
short	intent_code	68B	2B	nifti intent.
short	datatype	70B	2B	Data type.
short	bitpix	72B	2B	Number of bits per voxel.
short	slice_start	74B	2B	First slice index.
float	pixdim[8]	76B	32B	Grid spacings (unit per dimension).
float	vox_offset	108B	4B	Offset into a .nii file.
float	scl_slope	112B	4B	Data scaling, slope.
float	scl_inter	116B	4B	Data scaling, offset.
short	slice_end	120B	2B	Last slice index.
char	slice_code	122B	1B	Slice timing order.
char	xyzt_units	123B	1B	Units of pixdim[1..4].
float	cal_max	124B	4B	Maximum display intensity.
float	cal_min	128B	4B	Minimum display intensity.
float	slice_duration	132B	4B	Time for one slice.
float	toffset	136B	4B	Time axis shift.
int	glmax	140B	4B	Not used; compatibility with analyze.
int	glmin	144B	4B	Not used; compatibility with analyze.
char	descrip[80]	148B	80B	Any text.
char	aux_file[24]	228B	24B	Auxiliary filename.
short	qform_code	252B	2B	Use the quaternion fields.
short	sform_code	254B	2B	Use of the affine fields.
float	quatern_b	256B	4B	Quaternion b parameter.
float	quatern_c	260B	4B	Quaternion c parameter.
float	quatern_d	264B	4B	Quaternion d parameter.
float	qoffset_x	268B	4B	Quaternion x shift.
float	qoffset_y	272B	4B	Quaternion y shift.
float	qoffset_z	276B	4B	Quaternion z shift.
float	srow_x[4]	280B	16B	1st row affine transform
float	srow_y[4]	296B	16B	2nd row affine transform.
float	srow_z[4]	312B	16B	3rd row affine transform.
char	intent_name[16]	328B	16B	Name or meaning of the data.
char	magic[4]	344B	4B	Magic string.

Table A.1: Fields of NIfTI-1 Format Header, from *https://brainder.org/2012/09/23/the-nifti-file-format/*

B

Flowchart of Functions in FIONA

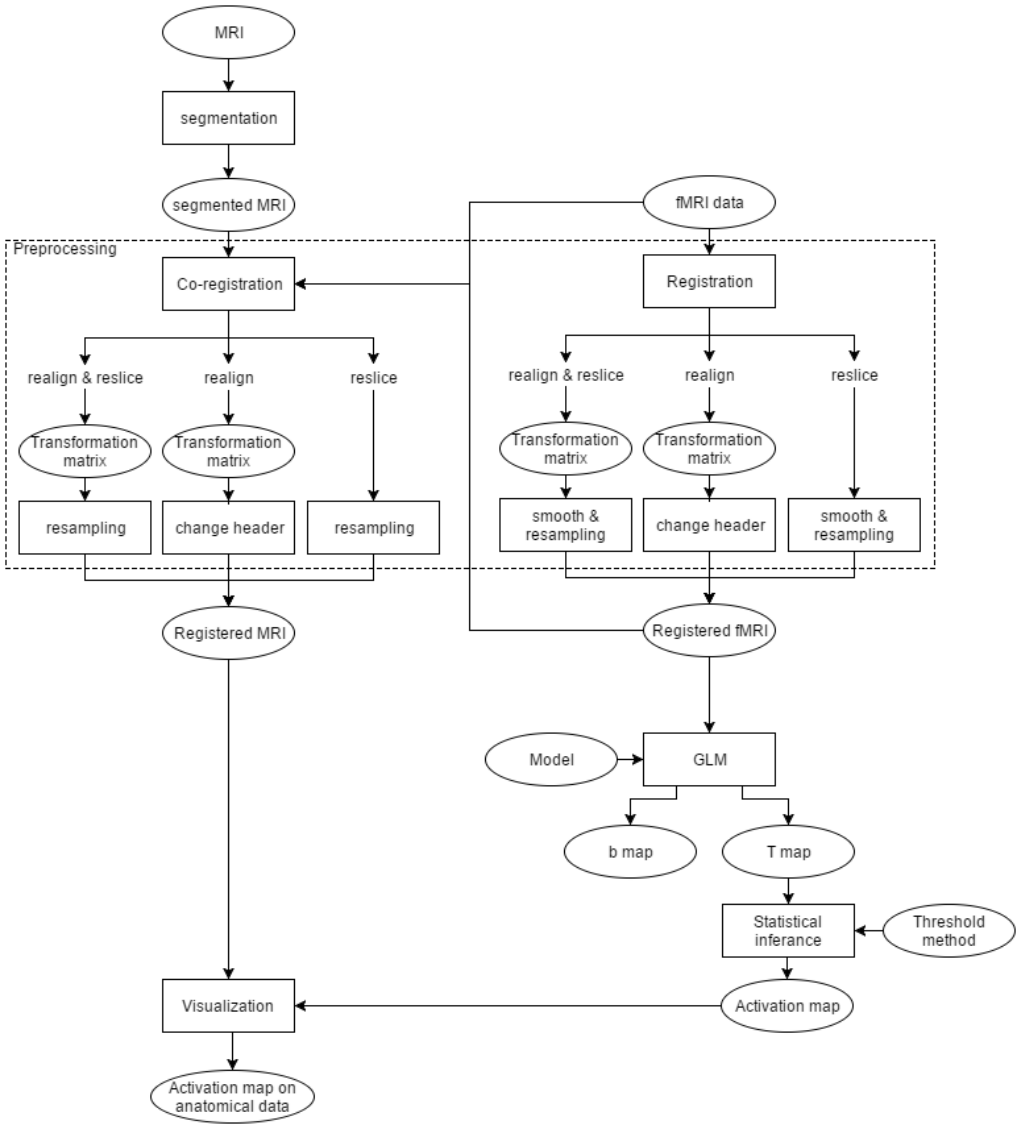

Figure B.1: A flowchart of the pipeline is made to clarify the information flow and functions incorporated into the software.

C

Parameters Details in Optimizer Benchmarking

- **ITK::RegularStepGradientDescentOptimizerv4**
  Learning rate to 1 as default;
  Relaxation factor to 0.5 as default;
  Maximum number of iterations to 500;
  Minimum step length to 10^{-6};
  Parameter scales to [1, 1, 1, 10^{-3}, 10^{-3}, 10^{-3}];
  Multi-resolution level to 3;
  Multi-resolution shrink factors to [1, 1, 2];
  Multi-resolution smoothing factors to [0, 1, 2];

- **ITK::GradientDescentLineSearchOptimizerv4**
  Learning rate to be estimated at each iteration;
  Maximum number of iterations to 500;
  Minimum convergence value to 10^{-6} as suggested;
  ϵ (Accuracy of line search) to 0.001;
  Upper limit to 2 as default;
  Lower limit to 0 as default;
  Convergence window size to 50 to pass all tests;
  Parameter scales to [1, 1, 1, 10^{-3}, 10^{-3}, 10^{-3}];
  Multi-resolution level to 3;
  Multi-resolution shrink factors to [1, 1, 2];
  Multi-resolution smoothing factors to [0, 1, 2];

- **ITK::ConjugateGradientLineSearchOptimizerv4**
  All setting to the same as in ITK::GradientDescentLineSearchOptimizerv4;

- **ITK::QuasiNewtonOptimizerv4**
  Learning rate to be estimated at each iteration;
  Maximum number of iterations to 500;
  Minimum convergence value to 10^{-6} as suggested;
  Convergence window size to 50 to pass all tests;
  Parameter scales to be estimated by ITK::RegistrationParameterScalesFromPhysicalShift<Metric>;
  Multi-resolution level to 3;
  Multi-resolution shrink factors to [1, 1, 2];
  Multi-resolution smoothing factors to [0, 1, 2];

- **ITK::AmoebaOptimizerv4**
  Initial simplex to be generated automatically;
  Maximum number of iterations to 500;

Parameter convergence tolerance to 10^{-6};
Function convergence tolerance to 10^{-6};
Parameter scales to [1, 1, 1, 10^{-3}, 10^{-3}, 10^{-3}];
Multi-resolution level to 3;
Multi-resolution shrink factors to [1, 1, 2];
Multi-resolution smoothing factors to [0, 1, 2];

- **ITK::OnePlusOneEvolutionaryOptimizerv4**
  Normal variate generator generated by ITK::Statistics::NormalVariateGenerator;
  Initial radius to 0.625 according to *registration.optimizer.OnePlusOneEvolutionary* class in MATLAB;
  Grow factor to 1.05 according to *registration.optimizer.OnePlusOneEvolutionary* class in MATLAB;
  Shrink factor to 0.95 according to *Martin Styner*;
  ϵ (Minimum radius) to 10^{-4};
  Maximum number of iterations to 5000;
  Multi-resolution level to 1 (Multi-resolution not applied).

Bibliography

[1] F. Gregory Ashby. *Statistical Analysis of fMRI Data*. The MIT Press, 2011. ISBN 978-0-262-01504-2.

[2] Barker AT, Jalinous R, and Freeston IL. Non-invasive magnetic stimulation of human motor cortex. *Lancet (London England)*, 325(8437):1106–1107, 1985.

[3] Yoav Benjamini and Joseph Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society, Series B*, 57(1):289–300, 1995.

[4] Michael A. Dimyan and Leonardo G. Cohen. Contribution of transcranial magnetic stimulation to the understanding of mechanisms of functional recovery after stroke. *Neurorehabil Neural Repair*, 24(2): 125–135, 2010.

[5] Fletcher Dunn and Ian Parberry. *3D Math Primer for Graphics and Game Development*. Wordware Publishing, Inc, 2002. ISBN 1-55622-911-9.

[6] Luis Ibanez and William Schroeder. *The ITK Software Guide*. Kitware, Inc., 2015. ISBN 978-1930934276.

[7] Ashburner J and Friston KJ. Nonlinear spatial normalization using basis functions. *Human Brain Mapping*, 7(4):254–266, 1999.

[8] Mark Jenkinson and Stephen Smith. A global optimisation method for robust affine registration of brain images. *Medical Image Analysis*, 5:143–156, 2001.

[9] O'Reardon JP, Solvason HB, Janicak PG, Sampson S, Isenberg KE, Nahas Z, McDonald WM, Avery D, Fitzgerald PB, Loo C, Demitrack MA, George MS, and Sackeim HA. Efficacy and safety of transcranial magnetic stimulation in the acute treatment of major depression: a multisite randomized controlled trial. *Biological Phychiatry*, 62(11):1208–1216, 2007.

[10] Friston KJ. Models of brain function in neuroimaging. *Annual review of psychology*, 56:57–87, 2005.

[11] Stefan Klein, Marius Staring, and Josien P. W. Pluim. Evaluation of optimization methods for nonrigid medical image registration using mutual information and b-splines. *IEEE Transaction on Image Processing*, 16(12):2879–2890, 2007.

[12] Jean-Pascal Lefaucheur, Nathalie André-Obadia, Andrea Antal, Samar S. Ayache, and Chris Baeken. Evidence-based guidelines on the therapeutic use of repetitive transcranial magnetic stimulation (rtms). *Clinical Neurophysiology*, 125(11):2150–2206, 2014.

[13] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. *Proceeding DARPA Image Understanding Workshop*, pages 121–130, 1981.

[14] David Mattes, David R. Haynor, Hubert Vesselle, Thomas K. Lewellen, and William Eubank. Pet-ct image registration in the chest using free-form deformations. *IEEE Transaction on Medical Imaging*, 22(1): 120–128, 2003.

[15] S.F.W. Neggers, T.R. Langerak, D.J.L.G. Schutter, R.C.W. Mandl, Ramsey NF, Lemmens PJ, and Postma A. A stereotactic method for image-guided transcranial magnetic stimulation validated with fmri and motor-evoked potentials. *NeuroImage*, 21(4):1805 –1817, 2004.

[16] Yan Niu, Zhiwen Xu, and Xiangjiu Che. Dynamically removing false features in pyramidal lucas–kanade registration. *IEEE Transaction on Image Processing*, 23(8):3535–3544, 2014.

[17] Fitzgerald PB, McQueen S Hoy K, Maller JJ, Herring S, Segrave R, Bailey M, Been G, Kulkarni J, and Daskalakis ZJ. A randomized trial of rtms targeted with mri based neuro-navigation in treatment-resistant depression. *Neuropsychopharmacology*, 34(5):1255–1262, 2009.

[18] Josien P. W. Pluim, J. B. Antoine Maintz, and Max A. Viergever. Mutual-information-based registration of medical images: A survey. *IEEE Transaction on Medical Imaging*, 22(8):986–1004, 2003.

[19] D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24(111), 1970.

[20] Martin Styner, Christian Brechb ühler, Gàabor Szèkely, and Guido Gerig. Parametric estimate of intensity inhomogeneities applied to mri. *IEEE Transaction on Medical Imaging*, 19(3):153–165, 2000.