Westfälische Hochschule
Gelsenkirchen Bocholt Recklinghausen
University of Applied Sciences

FACHBEREICH ELEKTROTECHNIK UND ANGEWANDTE NATURWISSENSCHAFTEN
ABTEILUNG MOLEKULARE BIOLOGIE

Development of an Application for MEG/MRI Co-Registration

Entwicklung einer Anwendung zur MEG/MRT-Co-Registrierung

Bachelorarbeit

im Rahmen der Erlangung des akademischen Grades
Bachelor of Sciences (B.Sc.)
im Studiengang Molekulare Biologie
im Schwerpunkt Bioinformatik

vorlegt von
Philo Reipke

Matrikelnummer:
201122106

Recklinghausen, August 2014

Angaben über Arbeitsgruppe und Betreuer

Diese Arbeit wurde durchgeführt am

Institut für Biomagnetismus und Biosignalanalyse

der Westfälischen Wilhelms-Universität Münster

unter der Anleitung von Priv.-Doz. Dr Carsten Wolters

Betreuer:

1. Gutachter: Prof. Dr. Heinrich Brinck

2. Gutachter: Priv.-Doz. Dr. Carsten Wolters

Westfälische Hochschule

Fachbereich Elektrotechnik und angewandte Naturwissenschaften

Abteilung Molekulare Biologie

August-Schmidt-Ring 10

45665 Recklinghausen

Erklärung zur Bachelorarbeit

Name:	Philo Reipke
Matrikelnummer:	201122106
Westfälische Hochschule Gelsenkirchen Bocholt Recklinghausen	Abteilung Recklinghausen
Fachbereich	Elektrotechnik u. angewandte Naturwissenschaften

Hiermit erkläre ich, dass ich die Bachelorarbeit

Development of an Application for MEG-MRI Co-Registration

Entwicklung einer Anwendung zur MEG-MRT Co-Registrierung

selbständig verfasst, nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

.....

Datum Unterschrift

Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich während meines Projekts und bei der Anfertigung dieser Arbeit unterstützt haben.

Ein besonderer Dank geht dabei an Priv.-Doz. Dr. Carsten Wolters und Andreas Wollbrink für die Anleitung der Projektarbeit und Betreuung am Institut für Biomagnetismus und Biosignalanalyse in Münster sowie an Prof. Dr. Heinrich Brinck der Westfälischen Hochschule für die hervorragende Betreuung und Unterstützung während meines Projekts und der Anfertigung dieser Arbeit.

Darüber hinaus bedanke ich mich bei Ümit Aydin, Ute Trompeter und allen Mitarbeitern des Instituts für Biomagnetismus und Biosignalanalyse sowie bei meiner Kommilitonin und Projektpartnerin Marie Theiß, die mich mit der Aufnahme und Verarbeitung der verwendeten Test-Datensätze unterstützt und die Entwicklung meiner Anwendung vorangebracht haben.

Ein Dank gilt auch an Stefan Neumann, der mir, durch die Möglichkeit seinen ehemaligen Kurs Scientific Computing zu besuchen, mein Interesse für die Programmierung wieder geweckt und meine Neuausrichtung hin zur naturwissenschaftlichen Informatik maßgeblich beeinflusst hat.

Zusammenfassung

Die vorliegende Arbeit befasst sich mit der Entwicklung einer Anwendung zur MEG/MRT-Co-Registrierung unter Verwendung der Plattform Java. Ziel dabei war es eine Anwendung zu entwickeln, welche die Möglichkeit zur Registrierung von Punktwolken an polygonale Kopfoberflächen mittels des ICP-Algorithmus bietet, der durch rigide Transformation die Distanz benachbarter Punktepaare zwischen den Datensätzen minimiert. Die Anwendung sollte sich im Gegensatz zu etablierten Anwendungen auf möglichst vielen Betriebssystem ohne das Vorhandensein zusätzlicher Software ausführen lassen und durch einfache Verständlichkeit und Bedienung der Programmoberfläche auszeichnen. Die angewandte Methodik sollte dabei eine Optimierung zur manuellen Vorgehensweise zur MEG/MRT-Co-Registrierung bieten und qualitativ gleichwertige oder optimierte Ergebnisse im Vergleich zu etablierten Anwendungen liefern. Unter Verwendung vorhandener Testdaten konnte gezeigt werden, dass die geschaffene Anwendung nachvollziehbare und reproduzierbare Ergebnisse liefert. Ob die erzielten Transformations-Ergebnisse eine optimierte Lokalisierung von kortikaler, neuronaler Aktivität ermöglichen, ist durch die Einbeziehung der Anwendung in den allgemeinen Arbeitsablauf noch nachzuweisen.

Stichwörter

MEG-MRT, Co-Registrierung, Polhemus Digitizer, Iterative Closest Point, ICP, rigide Transformation, Surface Registration, Java, Anwendung, JSurfReg

Abstract

This thesis is about the development of an application for MEG/MRI-co-registration using the Java platform. The aim was to develop an application that allows the visualization and processing of a given set of points by registering it to a given polygonal surface representation, using the iterative closest point algorithm to perform a rigid transformation that minimizes the mean distance between pairs of closest points among both data sets. In contrast to established applications, this new application was meant to be easy to deploy on almost any operating system without the need of further software and to provide an intuitive and easy-to-use graphical user interface (GUI). These goals have been accomplished completely. The processing of first data sets provided comprehensible and reproducible results that have to be involved in the process of localizing cortical neuronal activity in future to evaluate their quality.

Keywords

MEG-MRI, co-registration, Polhemus digitizer, tterative closest point, ICP, surface registration, rigid transformation, Java, application, JSurfReg

Contents

Abbreviations

CT	X-ray computed tomography
CTF	CTF MEG head coordinate system, same as SCS
EEG	Electroencephalography
FIDENTIS	Forensic 3D Facial Identification Software
GUI	Graphical user interface
ICP	Iterative closest point (algorithm)
JAMA	Java Matrix Package
JRE	Java runtime environment
JVM	Java virtual machine
LPA	Left peri-auricular
MEG	Magnetoencephalography
MRI	Magnetic resonance imaging
NAS	Nasion
OBJ	Wavefront OBJECT file format
POS	EEG POS file format
RPA	Right peri-auricular
SCS	Subject coordinate system, same as CTF

Figures

1 Introduction

In medical diagnostics neuronal imaging techniques such as Electroencephalography (EEG) or Magnetoencephalography (MEG) occupy an inferior role compared to the widely established imaging techniques such as Magnetic resonance imaging (MRI) or computed tomography (CT), yet their role is of great importance for research and therapy in some specific areas, e.g. in the field of epilepsy.

The strength of neuronal imaging techniques is the ability of capturing electro-magnetic fields caused by neuronal activity inside our brains, generated by thoughts, motions and visceral functions controlled by our autonomic nervous system.

Both systems, EEG and MEG, thereby deliver a great temporal resolution, yet their spatial resolution is imprecise. By combining EEG and MEG the spatial resolution can be improved. In the field of epilepsy, an improvement of the spatial resolution is greatly appreciated. Epilepsy is caused by degenerated neurons. These neurons show a permanently increased activity that can be measured by neuronal imaging techniques. Finding the accurate location of the cause of epileptic symptoms is of great importance for therapy, e.g. to prepare a surgery to remove the damaged tissue.

To locate neuronal activity, the data captured by neuronal imaging techniques has to be combined with anatomical data acquired using medical imaging techniques such as MRI, allowing the evaluation of neuronal activity according to the individual anatomy. The process of combining both types of data is referred to as MEG-MRI co-registration, allowing the location of neuronal activity within a given area. The size of this area depends very much on the spatial resolution of the used neuronal imaging technique, but also on the resolution of the MRI data and the quality of the performed MEG-MRI co-registration.

As for co-registration, there exist many different approaches to perform the combination of neuronal and anatomical imaging data. They all have in common that the positions of the head localization coils, coils attached to the subject's head during acquiring MEG data, have to be recovered within the anatomical imaging data to perform a data transformation.

1.1 Aim

Aim of this thesis is to develop a new application for MEG-MRI co-registration, applying the methodology of employing an additional head surface representation that has been digitized by using a Polhemus FASTRAK system. This set of digitized points will be matched to a polygonal head surface representation by using the iterative closest point (ICP) algorithm. The applied transformation during this surface registration will be stored to allow the exportation for further usage.

The developed application must thereby meet certain criteria that ensure reliability and the outcome of an alternative for MEG-MRI co-registration to established tools and software such as FieldTrip (see [1]) or MNE (see [2]). It must be easy to deploy on the most common operating system (e.g. Windows, Linux, Mac) without the need of making any adjustments or installing of further software. The provided GUI must be intuitive and easy to handle and must not require any programming skills to perform the surface registration.

The quality and accuracy of the achieved results applied by the used methodology will not be evaluated in this thesis, but will be discussed at the end, also regarding a sample transformation obtained by using a FieldTrip workflow.

1.2 Thesis Structure

The first chapter of this thesis is providing a short introduction to the field of neuroimaging, stating the motivation behind the aim of developing a new application. The second chapter explains theory necessary to understand the topic of MEG-MRI co-registration and goes into detail about the methodical approach that will be applied by the developed application. The third chapter illustrates the design of the application. It describes the supported file formats and goes into detail about the implemented functions and algorithms, clarifying how the transformation for MEG-MRI co-registration is being computed, providing pseudocode to exemplify some parts of the algorithms. The fourth and last chapter reflects the aim and outcome of the developed application, summarizing features and discussing the applied methodology.

2 Theory and Methods

2.1 MEG-MRI Co-Registration

MEG-MRI co-registration is understood as the merging of MEG and MRI data necessary for the localization of brain activity due to transferring acquired MEG signals to a subject specific 3D head model constructed from MRI data.

When measuring MEG signals there are (usually three) head localization coils attached to a subject's skin. Depending on the used MEG system these localization coils will be employed to construct the coordinate system representing the acquired data (see 2.1.1 *CTF MEG Head Coordinate System*). Frequently measuring them allows the recovering of their position in real-time and enables the MEG system to compensate head position changes by adjusting the acquired data relative to the recovered positions of these head localization coils, or to be more precisely, by reconstruction the used coordinate system.

When combining acquired MEG with MRI data, the problem that both data types are stored with different coordinate system has to be solved as they are differing in the locations of their origin and the orientations of their (x, y and z) axes.

To receive both data types in the same coordinate system at least one data set has to be adjusted by performing a rigid transformation, a mathematical operation that preserves the distances between every pair of points within the given vector space, applying the same rotation and translation to every point stored within the object that is being transformed. This can usually be achieved by setting the positions of the head localization coils on the subject's 3D head model, computing the necessary transformation and applying it to the according MEG data. This step is referred to as MEG-MRI co-registration. Any inaccuracy within this step, e.g. by selecting the inexact positions of the head localization coils, might result in a significant deviance and thereby in a shift of the recovered locations for all MEG signals within the transformed data set.

The difference in the coordinate systems is being described in the subchapters below.

2.1.1 CTF MEG Head Coordinate System

The CTF MEG head coordinate system, also known as subject coordinate system (SCS), is commonly used to store MEG data. Its structure is based on the measured locations of three head localization coils (fiducials) that are attached to a subject's skin during the MEG data acquisition. These localization coils are frequently measured and allow to recover the position of the head inside the MEG helmet in real-time as the measured positions of the head localization coils are used to rebuild the coordinate system, minimizing the deviation of measured signal sources.

As described in the CTF MEG™ File Formats manual [3], the coordinate system is set up the following way:

> "The CTF MEG System sets the origin and axes orientation on a reference determined by the three head localization coils, shown in *Figure 1* as the green (left), red (right), and blue (nasion) dots.
>
> The origin is defined as the midpoint between the left and right preauricular fiducial points. This means the CTF MEG head coordinate system uses both positive (to the subject's left) and negative values (to the subject's right).

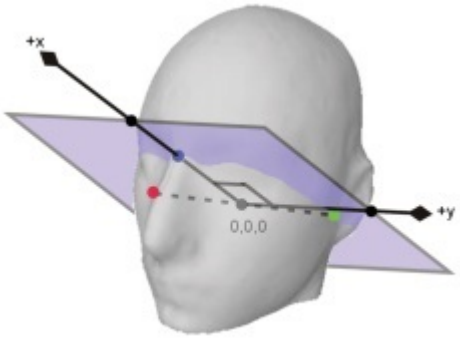

Figure 1: MEG references, origin and X-Y plane ([3], page 158)

> The orientation of the axes is determined by setting the x-axis on the line from the origin through the nasion fiducial point. The x-y plane (violet) is defined by the three fiducial points.

The y-axis is perpendicular to the x-axis on the x-y plane. Since the human head rarely has perfect symmetry, the y-axis is not likely through a fiducial point, but could be slightly ahead (as in this sample) or behind it.

The z-axis (see *Figure 2*) is perpendicular to both the x- and y-axes." ([3], page 157-159)

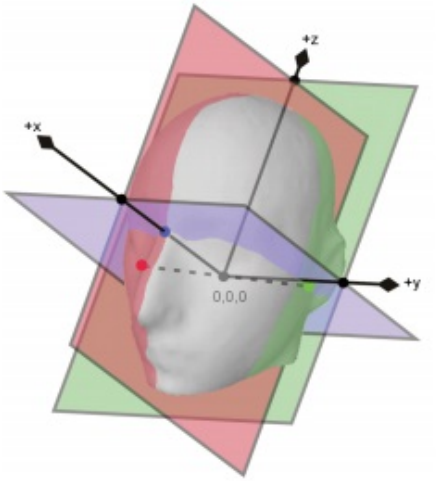

Figure 2: CTF MEG head coordinate system ([3], page 159)

2.1.2 MRI Voxel Coordinate System

Medical imaging methods such as MRI produce regular, rectangular arrays of points and cells. The points stored within these arrays is referred to as voxels, volumetric pixels, storing the actual imaging data. These voxels are separated by a spacing, e.g. 2 mm, depending on the resolution of the used MRI system. The total number of voxels produced depends on the imaging size and settings of the used MRI system, e.g. 256x256x256 voxels. The coordinate system that represents these voxels is thereby referred to as MRI voxel coordinate system or image coordinate system. Its origin is often located in the upper left corner, their x axis pointing to the right, their y axis pointing to the bottom and their z axis pointing backwards (DICOM standard, see [4]). The coordinate system's origin is equivalent to the first voxel stored, its index to be 1/1/1.

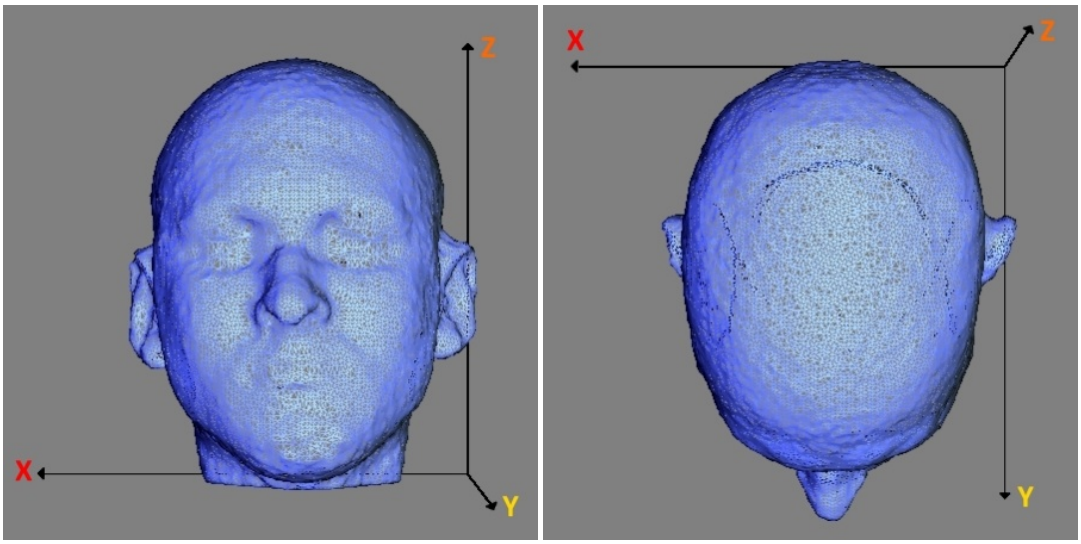

Figure 3: Meshed head surface with representation of its X, Y and Z axes.

By using the imaging data stored within these voxels, an image-segmentation to create 3D head models can be performed. If this results in the change of the initial coordinate system depends very much on the software used for segmentation. *Figure 3* shows an example for a 3D head model and the coordinate system representing the polygonal data where the software used to segment the MRI image data generated a new coordinate system.

2.2 Polhemus FASTRAK System

In chapter 2.1 the topic of MEG-MRI co-registration has been described, giving an idea how it can be performed using an easy approach by selecting the positions of the three head localization coils on a 3D head surface model followed by computing the necessary coordinate transformation. As the selected positions will rarely be precise, this will most likely result in a significant deviation and a shift of the localized source for the acquired MEG signals. To minimize this deviation, the developed application is going to use a different approach, supporting the use of an additional head surface representation that has been acquired using a Polhemus FASTRAK® system.

The Polhemus FASTTRAK system is a 3D digitizer and motion tracker with high resolution, accuracy and range. It can be applied for high accuracy head tracking or the digitization of head surfaces. The system can be set to use the MEG CTF coordinate system. By recording

the exact positions of the three head localization coils attached to a subject's skin before or after acquiring MEG data, the Polhemus system can set up the exact same coordinate system as the MEG is using to store its acquired data. Recording additional points representing the scalp and characteristic facial features (e.g. about 100 points) then allows the possession of a head surface representation using the CTF MEG head coordinate system. This digitized surface representation can be used to significantly improve the quality of the MEG-MRI co-registration.

Reflecting the purpose of the co-registration, its goal is to match MEG and MRI data. Instead of recovering the exact positions of the attached head localization coils on a 3D head surface or within MRI images, the digitized surface representation acquired with the Polhemus system can now be used to perform the registration. By using the ICP Algorithm (see 2.3 *Iterative Closest Point Algorithm*) the digitized head surface representation (acquired using Polhemus FASTRAK) can be easily registered to a polygonal 3D model (made by segmenting MRI data) representing the same head surface.

Using this method, manually selecting the positions of the head localization coils can be avoided, potentially resulting in achieving a minimization of the deviation and a more accurate localization of neuronal activity.

2.3 Iterative Closest Point Algorithm

The Iterative Closest Point (ICP) [5] algorithm used in the described approach for MEG-MRI co-registration was developed by Besl and McKay in the year 1992. The algorithm is reducing the general not-linear minimization problem of matching shape data of any kind to an iterative point matching problem.

Given two data sets consisting of point coordinates representing two 3D model shapes that may correspond to one another, the iterative closest point algorithm delivers an approach to estimate the optimal rotation and translation that aligns or registers both shapes by minimizing the distance between them, allowing determination of the equivalence of the shapes via a mean-square distance metric.

The algorithm can be initialized by forwarding two data sets, indicated as target $\boldsymbol{A} = \{\boldsymbol{P_1}, \boldsymbol{P_2}, \ldots, \boldsymbol{P_i}\}$ and source $\boldsymbol{B} = \{\boldsymbol{P_1}, \boldsymbol{P_2}, \ldots, \boldsymbol{P_i}\}$. Using an iterative process the algorithm registers the source data set $\boldsymbol{B}$ to the target data set $\boldsymbol{A}$. Employing the Euclidean distance the algorithm finds the closest neighboring point in data set $\boldsymbol{A}$ for each point in data set $\boldsymbol{B}$. The resulting mean squared distance in between all pairs of points has now to be minimized by determining a specific translation and rotation. The resulting transformation parameters will then be used to transform source data set $\boldsymbol{B}$. The iteration will be repeated until no significant change of the mean squared distance between the pairs of closest points can be achieved (threshold).

The following chapters are going into more detail about the ICP algorithm and the computations made to find the ideal transformation parameters as the implemented algorithm is being described (see 3.3.2 *Iterative Closest Point Algorithm*).

3 Implementation

3.1 Java Programming Language

The first step in developing the application was to choose a programming language. It was decided to use the object-oriented programming language Java that is the most commonly used programming language today, being present on over a billion different devices (desktop computers, laptops, tablets, mobile phones, ...) and thereby being highly established.

Java provides a great portability, induced by the fact that Java source code compiles to bytecode (intermediate code). This intermediate code can be run on any Java virtual machine (JVM) available for many different operating system (Windows, Linux, Mac OS), making it so successful. Java also delivers various different interfaces and frameworks to work with languages such as PHP, Python, Ruby or JavaScript, giving developers the chance to integrate functions written in other languages into their applications.

The recent major release of Java Platform, Standard Edition 8 and its JDK (see [6]) included Java FX 3D that brought new graphics features to Java without the need of using external graphic libraries such as OpenGL. These features are used to generate and display 3D objects within the developed application. The platform Java FX also allows application developers to easily create applications that behave consistently across multiple platforms, thereby fitting the set requirements perfectly.

Being the most common programming language also gave the opportunity to benefit from a large number of open source projects and libraries, such as JAMA (see 3.2 *JAMA*) or FIDENTIS (see 3.3 *FIDENTIS*) that assisted the development of the application.

3.2 JAMA

JAMA (see [7]) is an open source basic linear algebra package for Java that provides user-level classes for constructing and manipulating real, dense matrices. The JAMA classes *Matrix* and *EigenvalueDecomposition* have been integrated completely to the developed application and are used to perform matrix operations necessary for the implementation of the ICP algorithm (see 3.3.2*: Iterative Closest Point Algorithm*).

3.3 FIDENTIS

Forensic 3D Facial Identification Software (FIDENTIS, see [8]) is an open source project based on multidisciplinary cooperation with the goal to do research in the field related to 3D representations of the human face and to develop software solutions for it. The application is being developed by Faculty of Informatics of Masaryk University (MU) in Brno to be used in research of Department of Anthropology at Faculty of Science MU. The FIDENTIS source code provided implementations of a k-d tree and the ICP algorithm used within the developed application. The subchapters below are giving a detailed description to both of them.

3.3.1 K-D Tree

A k-dimensional tree is a data structure that can be used to organize a set of points, doing so by partitioning the data set's k-dimensional space, using keys to access the separated dimensions. To apply a k-d tree is useful when performing searches within multi-dimensional data, achieving a much faster computation time than with any linear methods.

The k-d tree used within the developed application was provided by the FIDENTIS source code and represents a three dimensional structure.

The basic structure of the k-d tree is an object of type *TreeMap* (see [9]), a Java implementation for a balanced red-black tree. It represents the first dimension (X) and is used to create nodes for every X coordinate in space. When adding a point, the algorithm tries to find a present node, using the specific X coordinate as the key value. If it is not to be found, a new node for this coordinate will be stored, therefore creating a new *TreeMap*. This new tree will represent the Y coordinates having the same X coordinates. If now again no object using the specific key value (Y) is present, a new object of type *Set* (see [9]) will be created. This object represents a list, storing all Z coordinates that have the same X and Y coordinates as its previous nodes.

The pseudocode below exemplifies the implemented method for adding points to the k-d tree.

```
Class: KdTree
Method: put
Input: Point3D (X, Y, Z)

TreeMap<Key, TreeMap<Key, Set<Key>>> TREE
TREE = new TreeMap ()

IF (TREE containsKey X) THEN          // searching first dimension for x key
    IF (TREE.getBranch(X) containsKey Y) THEN          // search for y key
        TREE.getBranch(X).getSet(Y).add(Z)             // add key for z
    ELSE
        Set<Key> setY = new Set()      // storing points with same x and y key
        setY.add(Z)                    // add key for z
        TREE.getBranch(X).addSet(Y, setY)
    ENDIF
ELSE
    TreeMap<Key, Set<Key>> branchY = new TreeMap()
    Set<Key> setY = new Set()          // storing points with same x and y key
    setZ.add(Z)                        // add key for z
    branchY.addSet(Y, setY)            // add Set to tree for specific key
    TREE.putBranch(X, branchY)         // contains x key
ENDIF
```

Given a source point, the closest target point within a given k-d tree can be found by performing a range check. This range check uses the source point's values as keys and adds an error range to search all three dimensions, finding all target points within a given key range defined as $\boldsymbol{range = key \pm error}$. All found points will be added to a list. The closest target point can then be found by searching for the smallest value of the mean squared distance $\boldsymbol{d}$ between source and target points in space defined as:

$$\boldsymbol{d = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2 + (z_a - z_b)^2}} \quad \textbf{(1)}$$

3.3.2 Iterative Closest Point Algorithm

The implementation of the ICP algorithm used in the developed application was provided by the FIDENTIS source library. It requires for an input of two lists of points $\boldsymbol{A} = \{\boldsymbol{P_1}, \boldsymbol{P_2}, \ldots, \boldsymbol{P_i}\}$ and $\boldsymbol{B} = \{\boldsymbol{P_1}, \boldsymbol{P_2}, \ldots, \boldsymbol{P_i}\}$ with $\boldsymbol{P_i} = \{\boldsymbol{x_i}, \boldsymbol{y_i}, \boldsymbol{z_i}\}$. The first point list $\boldsymbol{A}$ represents the target object that will be partitioned by using a k-d tree. The second list $\boldsymbol{B}$ represents the source object that will be transformed. The algorithm is using a do-while loop, computing and performing a transformation of the source object in every iteration.

Every iteration begins by finding the closest target point $\boldsymbol{P_i} \in \boldsymbol{A}$ for every source point $\boldsymbol{P_j} \in \boldsymbol{B}$ by using a k-d tree, adding the found points to a new list $\boldsymbol{C} = \{\boldsymbol{P_1}, \boldsymbol{P_2}, \ldots, \boldsymbol{P_k}\}$.

The distance between every pair of points $\boldsymbol{P_j} \in \boldsymbol{B}$ and $\boldsymbol{P_k} \in \boldsymbol{C}$ with $\boldsymbol{j} = \boldsymbol{k}$ will thereby be stored, using it to compute the mean distance that we need to translate the source points. The pseudocode below illustrates the first part of the iteration:

```
Double xDistance                    // sums distances in x dimension
Double yDistance                    // sums distances in y dimension
Double zDistance                    // sums distances in z dimension
List<Vector3D> neighbors            // stores neighboring points

FOREVERY point IN source
    FIND CLOSTEST neighbor TO point IN target        // using k-d tree
    ADD neighbor TO neighbors
    ADD X DISTANCE FROM point TO neighbor TO xDistance
    ADD Y DISTANCE FROM point TO neighbor TO yDistance
    ADD Z DISTANCE FROM point TO neighbor TO zDistance
ENDFOR
```

After determining the closest target points $\boldsymbol{P_k} \in \boldsymbol{C}$ for every source point $\boldsymbol{P_j} \in \boldsymbol{B}$ and storing their mean distances for translation, the rotation has to be prepared.

The algorithm therefore computes a covariance matrix $\boldsymbol{M}$:

$$\boldsymbol{M} = \sum_{i=1}^{n} \boldsymbol{S_i^T T_i} \quad (2)$$

where $\boldsymbol{S_i}$ is defined as the matrix created from the specific source point $\boldsymbol{P_j} \in \boldsymbol{B}$ with $\boldsymbol{j} = \boldsymbol{i}$:

$$R_i = \begin{pmatrix} 0 & -p_x & -p_y & -p_z \\ p_x & 0 & p_z & -p_y \\ p_y & -p_z & 0 & p_x \\ p_z & p_y & -p_x & 0 \end{pmatrix} \quad (3)$$

and $\boldsymbol{T_i}$ to be defined as the matrix created from the closest target point $\boldsymbol{P_k} \in \boldsymbol{C}$ with $\boldsymbol{k} = \boldsymbol{i}$:

$$Q_i = \begin{pmatrix} 0 & -q_x & -q_y & -q_z \\ q_x & 0 & -q_z & q_y \\ q_y & q_z & 0 & -q_x \\ q_z & -q_y & q_x & 0 \end{pmatrix} \quad (4)$$

The pseudocode below illustrates the steps described above.

```
Matrix matrixM                                  // covariance matrix

FOREVERY point IN source PAIRED WITH neighbor IN neighbors
    CREATE matrixR FROM point                   // same index as neighbor
    CREATE matrixQ FROM neighbor                // same index as point
    TRANSPOSE matrixR
    MULTIPLY matrixR BY matrixQ
    SUM matrixQ TO matrixM
ENDFOR
```

The computed matrix $\boldsymbol{M}$ is then used to generate a quaternion for performing the rotation.

Quaternions are a number system that extends the complex numbers, obtaining an important role for performing rotations in $\mathbb{R}^3$.

A quaternion $\boldsymbol{q}$ can be declared as a 1x4 dimensional vector:

$$\boldsymbol{q} = \{\boldsymbol{w}, \boldsymbol{xi}, \boldsymbol{yj}, \boldsymbol{zk} | \boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{R}\} \quad \textbf{(5)}$$

To the complex numbers $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ thereby apply specific rules declared by Hamilton, e.g.:

$$\boldsymbol{i^2} = \boldsymbol{j^2} = \boldsymbol{k^2} = \boldsymbol{ijk} = \boldsymbol{-1} \quad \textbf{(6)}$$

$$\boldsymbol{ij} = \boldsymbol{k}, \quad \boldsymbol{jk} = \boldsymbol{i}, \quad \boldsymbol{ki} = \boldsymbol{j} \quad (7)$$

$$\boldsymbol{ji} = \boldsymbol{-k}, \quad \boldsymbol{kj} = \boldsymbol{-i}, \quad \boldsymbol{ik} = \boldsymbol{-j} \quad \textbf{(8)}$$

Regarding these complex numbers, quaternion multiplication is non-commutative and the multiplications thereby have to be performed in the right order.

The rotation quaternion $\boldsymbol{q} = \{\boldsymbol{q_w}, \boldsymbol{q_x}, \boldsymbol{q_y}, \boldsymbol{q_z}\}$ can be generated by using the eigenvector $\boldsymbol{e} = \{\boldsymbol{e_1}, \boldsymbol{q_2}, \boldsymbol{q_3}, \boldsymbol{q_4}\}$ representing the largest eigenvalue of the matrix $\boldsymbol{M}$.

Two quaternions $\boldsymbol{q} = \{\boldsymbol{w}, \boldsymbol{xi}, \boldsymbol{yj}, \boldsymbol{zk} | \boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{R}\}$ with $\boldsymbol{w} \neq \boldsymbol{0}$ can be multiplied to generate a new quaternion by using the Hamilton product**:**

$$\boldsymbol{q_{total}} = \boldsymbol{q_1 q_2} = (\boldsymbol{w_1} + \boldsymbol{x_1 i} + \boldsymbol{y_1 j} + \boldsymbol{z_1 k})(\boldsymbol{w_2} + \boldsymbol{x_2 i} + \boldsymbol{y_2 j} + \boldsymbol{z_2 k}) \quad \textbf{(8)}$$

Resulting in a new quaternion representing their combined rotation. To perform rotations with quaternions the following equation can be used:

$$\boldsymbol{r'} = \boldsymbol{qrq^{-1}} \quad \textbf{(10)}$$

with $\boldsymbol{r}$ to be the initial point $\vec{\boldsymbol{p}} = \{\vec{\boldsymbol{x}}, \vec{\boldsymbol{y}}, \vec{\boldsymbol{z}}\}$ written as a quaternion $\vec{\boldsymbol{r}} = \{\boldsymbol{w}, \vec{\boldsymbol{p}}\}$ with $\boldsymbol{w} = \boldsymbol{0}$ and $\boldsymbol{r'}$ to be representing the rotated quaternion. The point coordinates can be recovered by simply using the x, y and z values within the newly generated quaternion.

For more information about quaternions and how to perform rotations, refer to [13].

The multiplication using the same rotation quaternion $\boldsymbol{q}$ will be performed for all source points $\boldsymbol{P_j} \in \boldsymbol{B}$, also adding the mean distances in x, y and z distance that has been stored before to translate the source points, resulting in a completely transformed data set.

The iterative process stops as soon as a maximum number of iterations has been performed or the change in the Euclidean distance between the closest pairs of points drops below a threshold value.

3.4 File Formats

The developed application supports the input of two types of data, Wavefront OBJ and EEG POS files. Any data imported to the application will be displayed as one or several 3D objects in the viewer area.

More information about the supported file formats can be found in the subchapters below.

3.4.1 Wavefront OBJ

The Wavefront OBJ file format is a commonly used standard for the representation of polygonal data in ASCII form. It can contain various types of data, e.g. geometric vertices, texture vertices, vertex normals, points, lines or faces, each type of data being indicated by its own keyword, such as **v** for **vertex** or **f** for **face**. The most frequently encountered OBJ files contain data that represents polygonal faces. *Figure 6* shows an example for such a file that contains a square represented by two polygons (triangles).

```
# OBJ file containing content that represents a square

v 0 0 0
v 0 0 10
v 0 10 0
v 0 10 10

f 1 2 3
f 3 2 4
```

Figure 4: Example for an Wavefront OBJ (.obj) file.

The implemented method for reading Wavefront OBJ files is capable of reading any correctly formatted OBJ file. However, the data has to be representing a polygonal surface made from triangles. For constructing a polygonal surface, only vertex and face data has to be given. Any other information (i.e. textures, normals) within the files will be ignored.

As every triangle has a front and a back face and face normals are not being used to indicate which is which, the order of the vertices representing a triangle (face) is very important.

The developed application is using an object of type *TriangleMesh* (see [10]) provided by Java FX to display the surface within the viewer area, resulting in a polygonal surface representation that can be displayed in the form of a mesh or as a rendered surface. By default, this object is using the counter-clockwise (or right-hand rule) winding order to determine the front face. Using varying winding orders within the file might result in strange behavior of the displayed objects but should not affect the quality of the achieved transformation.

For more information refer to [10] for the *TriangleMesh* and to [11] for the OBJ file format.

3.4.2 EEG POS

The EEG POS file format is a tab-delimited ASCII text file that can be used by the Polhemus FASTRAK digitizing system to store the acquired data.

The first line of the text file always indicates the number of points stored within the file. The following lines then contain the actual data, each line representing a single point, beginning with the current point index (with 1 to be the first index) followed by its X, Y and Z coordinates, separated by a tab. Additionally, the file stores coordinates for the nasion, left and right ear coils at the end of the file. The lines then start with the keywords "nasion", "left" and "right", respectively. *Figure 3.3* shows an example for a typical POS file.

```
 124
 1          11.4538169952631      -0.641636243678181      1.12041521687021
 2          12.213724402466       -0.401580509654014      2.31460202385441
 3          12.7476969070662      -0.544275593263312      4.22765485695029
 ...
 122         9.17916288589873      0.657499599124094      -5.34742175496001
 123         9.20927125195794      1.47509154929974       -5.10159544216891
 124         8.67880214770184      1.87772891541176       -4.55622607371421
nasion      10.4303912840552       2.29850860566927       -3.76441770550545
left        -0.105092061810202     7.23306498089582        2.18114372049771
right        0.105092061810202    -7.23306498089582       -2.18114372049771
```

Figure 5: Example for an EEG POS (.pos) file.

The POS file data will be displayed within the applications viewer area by using predefined objects of type Sphere (refer to [10]). Every point within the data set is indicated by one sphere. The color and size of the points indication the positions of the three head localization (fiducials) coils differ from the other points stored within a single data set, i.e. regular points being blue and fiducials being purple.

For more information about the EEG POS file format, refer to [12].

3.5 Data Processing Workflow

The registration of surface (OBJ file) data and digitizer (POS file) is achieved in three steps, resulting in the output of a transformation matrix that can be stored in a ASCII text file by using the implemented export function.

The methods used to perform the data processing are described in the subchapters below.

3.5.1 Set Fiducials

After importing an OBJ file representing a polygonal head surface, the estimated positions of the head localization coils (also refered to as fiducials) have to be determined on the surface displayed within the viewer area.

By selecting the "Set fiducials" button below the viewer area, a new window will open. This window provides options to set and store the positions for the nasion (NAS), left (LPA) and right (RPA) head localization coils determined by clicking on the polygonal surface representation displayed in the viewer area.

The selected positions will be stored in the application kernel, represent as point data. This data is necessary for the following step, the pre-alignment of the POS file data to the OBJ file data described in the next subchapter.

3.5.2 Pre-Alignment

This data processing step requires a POS file to be imported to the application and the positions of the three head localization coils to be determined on the displayed polygonal head

surface representation. It is assuring that the imported POS file data and the OBJ file data share the same orientation regarding their represented head surfaces. An altered implementation of the regular ICP algorithm (see 3.3.2 *Iterative Closest Point Algorithm*) is used to perform this operation.

To initiate the algorithm, POS file data and the three point positions of the head localization coils selected in the previous step have to be forwarded. The POS file data will be used as source and the three forwarded positions will be used as target, thereby differing from the regular ICP algorithm that is using the OBJ file data as target. Also, the regular algorithm assigns pairs of points among source and target for every point within the source data set and uses them to compute the ideal translation and rotation. The altered algorithm is only using three fixed pairs of points to compute the transformation. Three specific points stored in the source data set, declared as "nasion" (NAS), "left" (LPA) and "right" (RPA) within the POS file (see 3.4.2) will be paired with one of the three forwarded points, each of them determined as one of the three specific points in the previous step.

The transformation parameters will still be computed as illustrated in the subchapter describing the implementation of the ICP algorithm (see 3.3.2) and will be applied to all points within the source data set. The algorithm iterates until the change of the mean distance between the paired points drops below the set threshold. The total amount of translation and rotation applied during this processing step is being stored.

An option to assign weightings to the each point representing the NAS, LPA or RPA point has been implemented. By assigning a specific weighting to any fiducial, the according pair of points will be involved to the computation of translation and rotation a specific number of times, resulting in a shift of the source data, bringing the related points closer together. The default setup assigns a weighting of 10 to the nasion, 1 to the left and 1 to the right fiducial. It can be changed by using the "Settings" menu item accessible in the "Edit" menu.

Figure 5 on the next page illustrates the results achieved by performing the pre-alignment. The purple spheres represent the NAS, LPA and RPA localization coils that are used for pre-aligning, the blue spheres represent all other points within the POS data set.

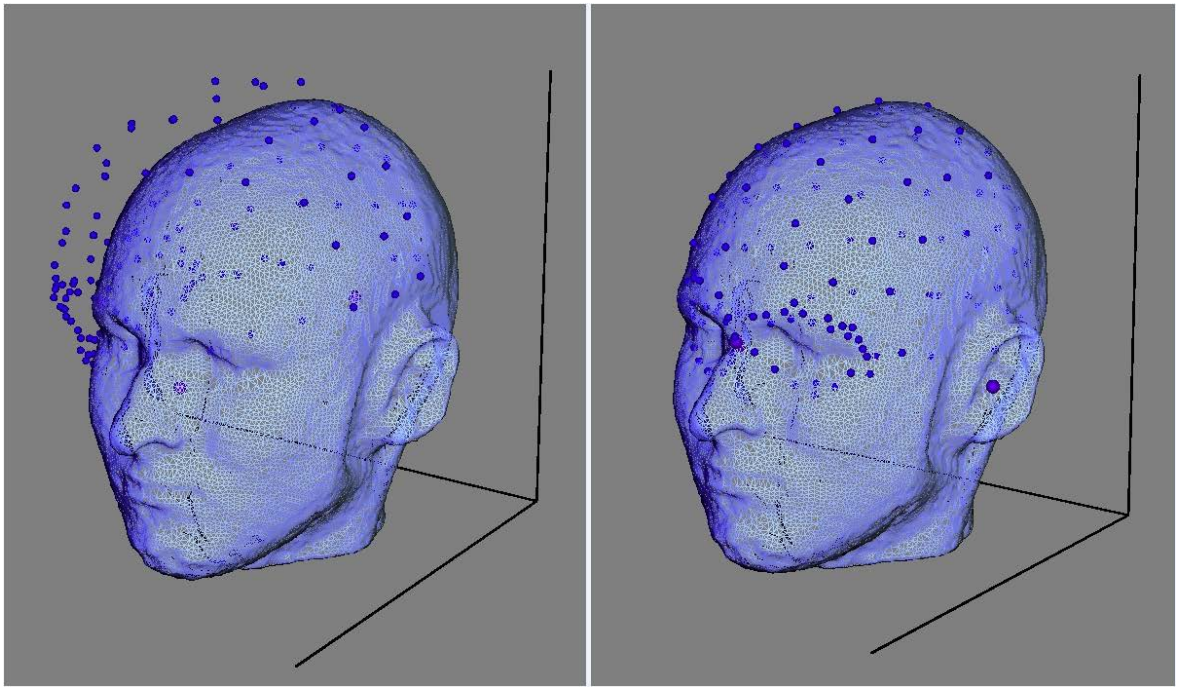

Figure 6: Comparison of POS file data orientation before (left) and after pre-aligning (right).

3.5.3 Registration

This step uses the regular ICP algorithm to perform the final registration of the pre-aligned POS file data to the current OBJ file data.

As described before (see 3.3.2), the algorithm requires for an input of two sets of points to initiate, using the POS file data as source and the OBJ file data as target. The algorithm then assigns pairs of closest points among source and target, using them to compute the transformation parameters that will be applied to all points within the source data set, iterating until the change of distance between the assigned pairs of points drops below the given threshold. The total amount of translation and rotation applied during this processing step is combined with the transformation applied in the previous processing step, the pre-alignment. The combined transformation is then being stored.

As for pre-aligning, the implementation for this algorithm is also providing options to apply weightings to the NAS, LPA or RPA points stored within the POS file data. By setting a value to 0, the specific point will not be involved to compute the transformation parameters at all.

The application is able to perform two registrations at once, using two different sets of options that allow the comparison of the achieved results with different parameters.

3.5.4 Transformation Export

A window showing the transformation applied to the POS file data during pre-aligning and registering can be opened by using the "Show transformation" button or by using the "Show > Transformation" menu item in the "File" menu. Additionally, the applied transformations can be exported to an ASCII text file by employing the implemented export function, accessible in the "File" menu by choosing the "Save > Transformation" menu item. A radio button menu will appear, providing options to either store the transformation applied during pre-alignment, registration #1 or registration #2, thereby referring to the two options sets available to perform the registration.

The rotation applied during pre-aligning and registration is stored in the form of quaternions. A normalized (unit) quaternion $\boldsymbol{q} = \{\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} | \boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{R}\}$ can be converted to a 3x3 dimensional rotation matrix $\boldsymbol{R}$ by using the following equation (from [5], 21):

$$\boldsymbol{R} = \begin{pmatrix} (q_w{}^2 + q_x{}^2 - q_y{}^2 - q_z{}^2) & 2(q_x q_y + q_z q_w) & 2(q_x q_z + q_y q_w) \\ 2(q_x q_y + q_z q_w) & (q_w{}^2 + q_y{}^2 - q_x{}^2 - q_z{}^2) & 2(q_y q_z + q_x q_w) \\ 2(q_x q_z + q_y q_w) & 2(q_y q_z + q_x q_w) & (q_w{}^2 + q_z{}^2 - q_x{}^2 - q_y{}^2) \end{pmatrix} \quad (11)$$

The translation applied during pre-alignment and registration is stored in the form of a point $\boldsymbol{P} = \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$, representing a vector from the coordinate system's origin to the specific point coordinates.

To export the transformation parameters, rotation and translation are combined into one 4x4 transformation matrix $\boldsymbol{T}$:

$$\mathbf{T} = \begin{pmatrix} q_w{}^2 + q_x{}^2 - q_y{}^2 - q_z{}^2 & 2(q_x q_y - q_w q_z) & 2(q_x q_z + q_w q_y) & x_{translate} \\ 2(q_x q_y + q_w q_z) & q_w{}^2 + q_y{}^2 - q_x{}^2 - q_z{}^2 & 2(q_y q_z + q_w q_x) & y_{translate} \\ 2(q_x q_z - q_w q_y) & 2(q_y q_z + q_w q_x) & q_w{}^2 + q_z{}^2 - q_x{}^2 - q_y{}^2 & z_{translate} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (12)$$

The transformation matrix gets stored in an ASCII text file, also containing information about the used OBJ and POS files. *Figure 7* below shows an example for a file generated using the export function.

```
Surface: A1047
Digitizer: A1457-EEG-151113

Mode: Pre-alignment

0.06447936411936683   -0.9970486597456039   -0.041669913633910205   92.8743748831943
0.9503489323661365   0.07409076956227878   -0.30223743086556026   113.08063379444243
0.3044327813381915   -0.020112880578493197   0.9523214550148118   83.25714418893509
0.0   0.0   0.0   1.0
```

Figure 7: Content of an ASCII text file generated by using the export function.

4 Summary and Discussion

4.1 Design

The aim of this thesis was to develop a reliable application that can be used for MEG-MRI co-registration as an alternative to established software and tools. The resulting application has been named *JSurfReg*, short for Java Surface Registration. It meets most of the criteria and requirements stated in the aim at the beginning of this thesis.

As Java is commonly present on most systems, *JSurfReg* provides a great portability and is easy to deploy. The implemented GUI adjusts its available options and displayed objects according to the present data and current processing step. It does not require any programming skills and thereby also provides less adjustable parameters than some other tools. The available options should still meet most users requirements. In advance to the implemented features provided by the newest version of Java FX, the GUI should behave consistently across multiple platforms. Still, its functionality has yet only been approved for Windows 7 (64 bit). However, it was also possible to run the application on an Ubuntu Linux (64 bit) operating system, but as the given system did not meet the hardware requirements set by Java FX 3D, trying to import files resulted in an error due to the fact that the systems graphic processing unit was not supported by the implemented Java FX 3D features.

4.1.1 Integration

Regarding the implemented file formats, it should be easy to integrate JSurfReg into most existing workflows.

When working with Polhemus systems, EEG POS files are a common standard and can be generated by the specific system. For the creation of polygonal surface representations, software tools often provide export functions to generate OBJ files. To convert data types, multiple software tools are available, as well as own methods for file conversion could be employed easily, e.g. by using MATLAB.

The output of an ASCII text file containing the achieved transformation allows easy conversion and use for further processing.

4.1.2 Data Processing

The JSurfReg workflow for data processing has been described in the subchapters of part 3.5. It contains three steps that perform actions analogous to established software and tools.

The first two processing steps, setting the head localization coils (fiducials) and pre-aligning the POS data, have to be performed to assure the right orientation of the current POS file data. This steps can be referred to as manual registration. It is an unavoidable step as the ICP algorithm always converges to a local minimum that differs regarding the initiate orientations.

The use of the estimated positions for the head localization coils on the polygonal head surface representation provides a comfortable and good method to obtain the right initial orientation. The option to assign weightings to each fiducial point allows to manipulate the pre-alignment. E.g. by assigning a weighting of 20 to the NAS point will almost fix it in place, not taking the positions of the LPA and RPA point into much account. For users that want to use the pre-alignment for manual alignment only, the export function allows storing of the applied transformation.

The final registration will then be performed by the ICP algorithm, minimizing the distances between both data sets. Assigning weightings to the fiducial points allows to make adjustments to the results. By providing the chance to perform the registration with two different options sets at once, users can compare the outcome of different registrations in the viewer area, deciding which one is of better quality. As for pre-aligning, every transformation can be stored easily by using the export function.

4.2 Methodology and Results

The methodology of registering digitized sets of points to polygonal head surface representations by using the ICP algorithm in terms of MEG-MRI co-registration has proven to be reliable in the past and has thereby been applied by other software and tools.

During the development of *JSurfReg*, two data sets, both consisting of a polygonal head surface representation and several digitized point clouds, have been available to evaluate first results achieved by using the applied methods.

The results shown in *Figure 8* indicate that the achieved transformation by using the ICP algorithm improved significantly compared to a manual registration (distance at 0 iterations) as the mean squared distance between the closest points among both data set have been minimized after about 15 to 25 iterations.

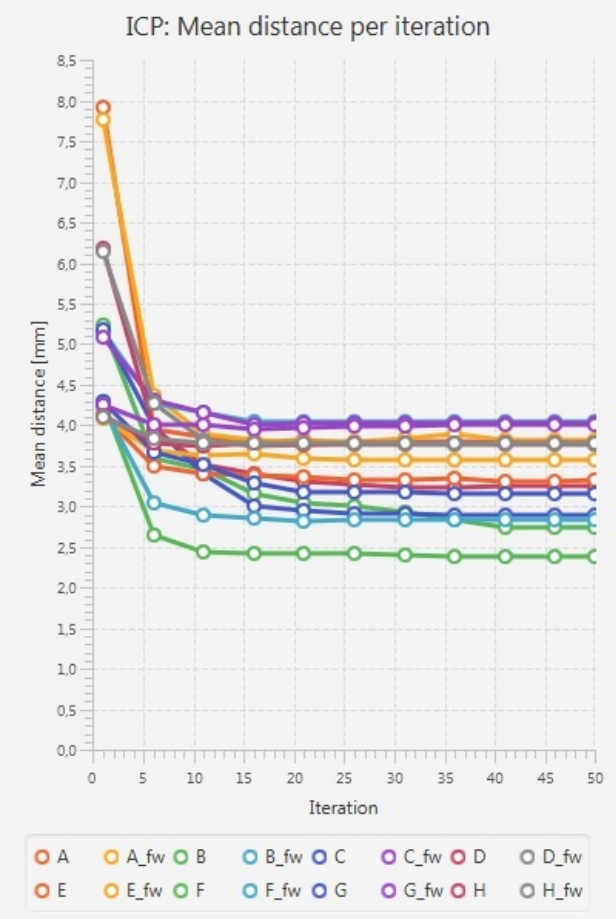

Figure 8: Surface registration, trend of the mean squared distance during iterations.

Still, in some cases, the registration appeared to be less accurate regarding the visualized positions of the head localization coils. Their new positions differed a lot from the previously set positions, moving them inside the polygonal head surface representation while all other points have been located on the surface. As the obtained positions of the head localization coils are of great importance (reminder: they are used for constructing the MEG coordinate system), this would most likely result in a significant shift of the projected localized sources for neuronal activity. To assign weightings, especially to the NAS point, improved the visual outcome of the registration but increased the mean squared distance.

This problem is considered to be caused by a deformation of the point sets acquired by the Polhemus FASTRAK digitizer system and has to be solved by improving the acquiring method. Also, the quality of the polygonal head surface achieved by segmenting the MRI data can cause a loss of accuracy. The higher the resolution of the acquired MRI data, the more precise can the segmentation be performed.

The results achieved with JSurfReg could not yet been compared into detail to results achieved with other software or tools. However, comparing a transformation matrix obtained by using a FieldTrip workflow compared to a transformation matrix obtained with JSurfReg for the same data was showing almost similar values. As the ICP algorithm always converges to a local minimum according to the initiate orientation, the obtained transformation matrix is not expected to be identical, hence the results seem to be very promising.

4.3 Outlook

It is expected that the developed application *JSurfReg* delivers a functionality similar to established software and tools and that the generated transformation results are reliable.

However, transformation results obtained by using JSurfReg should not be used for scientific or medical analysis and localization of neuronal activity before the expected reliability has been verified, for instance by trying to localize the source of signals acquired of test models.

The application can be further developed by implementing additional options, e.g. by giving the possibility of removing single points within the digitized point clouds or even by assigning weightings to points other than the fiducials. Also, an option to manually adjust the registration should be implemented.

In future, the application could be developed to provide additional functionalities or be combined with other software and tools. The availability of interfaces and frameworks to work with other programming languages such as Python, Ruby, JavaScript allows an easy way of implementing existing functions into Java applications.

Literature & References

[1] FieldTrip, Donders Institute for Brain, Cognition and Behaviour
URL: http://fieldtrip.fcdonders.nl/ (August 2014)

[2] M. S. Hämäläinen, MNE software, Athinoula A. Martinos Center for Biomedical Imaging
URL: https://martinos.org/mne/ (August 2014)

[3] VSM MedTech Ltd, CTF MEG Head Coordinate System, CTF MEG™ File Formats, page 157, Revision 1.1 (11 April 2006)

[4] Digitial Imaging and Communications in Medicine (DICOM), C.7.6.2.1.1 Image Position and Image Orientation, DICOM PS3.1
URL:http://medical.nema.org/dicom/2013/output/chtml/part03/sect_C.7.html#sect_C.7.6.2.1.1 (August 2014)

[5] Besl P.J., McKay N.D.: A method for Registration of 3D-Shapes, IEEE Trans PAMI, 14 (2), 1922

[6] Oracle, JDK 8 Release Notes
URL: http://www.oracle.com/technetwork/java/javase/8-relnotes-2226341.html (August 2014)

[7] JAMA: A Java Matrix Package
URL: http://math.nist.gov/javanumerics/jama/ (August 2014)

[8] FIDENTIS - Forensic 3D Facial Identification Software
URL: http://fidentis.cz/ (August 2014)

[9] Oracle, Java Platform SE 7 Documentation
URL: http://docs.oracle.com/javase/7/ (August 2014)

[10] Oracle, Java Platform SE 8 Documentation
URL: http://docs.oracle.com/javase/8/ (August 2014)

[11] FileFormat.info, Wavefront OBJ File Format
URL: http://www.fileformat.info/format/wavefrontobj/egff.htm (August 2014)

[12] VSM MedTech Ltd, EEG Pos File Format, CTF MEG™ File Formats, page 59, Revision 1.1 (11 April 2006)

[13] Berthold K.P. Horn: Closed-form solution of absolute orientation using unit quaternions, Journal of the Optical Society of America A, Vol. 4, page 629 (April 1987)

Appendix

All literature and references used in this thesis can be found on the attached CD-ROM. It also contains the current executable files for JSurfReg (v1.0), the relating source code and documentations (javadoc).