Evaluation of surface registration algorithms for PET motion correction

Bachelor thesis

Hans Martin Kjer and Jakob Wilm

Kongens Lyngby 2010
IMM-2010-9

Title:
Evaluation of surface registration algorithms for PET motion correction

Danish title:
Evaluering af overfladeregistreringsalgoritmer til PET bevægelseskorrektion

Authors:
Hans Martin Kjer, s072301 and Jakob Wilm, s072322

Supervisors:
Associate professor Rasmus R. Paulsen, DTU Informatics
Ph.D. student Oline V. Olesen, DTU Informatics

Handed in for evaluation on June 1st, 2010

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Summary

Recent advancements in high resolution Positron Emission Tomography (PET) have made it possible to study metabolic activity and the anatomy of the brain in great details. It becomes increasingly important to perform motion correction on these data. Motion tracking with a structured light system is pursued by Ph.D. student Oline V. Olesen.

This thesis covers the registration (alignment) of point cloud surfaces acquired from a structured light scanner.

An introduction to the concepts of rigid motion and point cloud properties is given. These include representations of rotations, surface curvature estimates and solutions to the nearest neighbor problem. The iterative closest point (ICP) algorithm is detailed with some of its most promising variants. Non linear optimization is introduced conceptually.

Experimental data is used to compare these different variants in terms of convergence, accuracy, robustness and speed.

It is concluded, that implementational simple variations of the ICP algorithm can yield much improved performance and reliability, and that, with the proper choice of acceleration techniques, real time performance is a realistic perspective.

In the appendix derivations of the most important mathematical formulas are given.

Resumé

De seneste års udvikling af højopløst Positron Emissions Tomografi (PET) har muliggjort studier af metabolisk aktivitet og detaljeret anatomi af hjernen. Dette gør det i stigende grad nødvendigt at foretage korrektion for patientens bevægelser under dataopsamling. Ph.D.-studerende Oline V. Olesen udvikler et bevægelsesregistreringssystem baseret på struktureret lys.

Nærværende opgave omhandler beregningen af den relative positionsforskel baseret på punkskyer fra et struktureret lys-system.

Opgavens første del giver en introduktion til rigide transformationer og punkskyers egenskaber. Disse inkluderer repræsentationer af en rotation i 3-D, overfladekurvatur og metoder til nærmeste nabo-søgninger. ICP-algoritmen *[eng: iterative closest point]* bliver beskrevet med nogle af dens varianter. Ikke-lineær optimering bliver beskrevet konceptuelt.

Ved hjælp af eksperimentelle data sammenlignes disse varianter med henblik på konvergens, nøjagtighed, robusthed og beregningstid.

Det konkluderes, at nogle simple varianter af ICP algoritmen kan øge hastighed og pålidelighed, og at realtidsberegninger er et realistisk perspektiv.

Udledninger af de mest centrale matematiske formler er givet i appendiks.

Acknowledgements

First of all we would like to thank our supervisor, associate professor Rasmus R. Paulsen. We greatly appreciate all the support and guidance we have received throughout this project. Be it helping with software issues, giving introductions to image analysis concepts or constructive feedback regarding the shaping of the project and the report.

This project is a spin-off from the Ph.D. project of Oline V. Olesen. We are very grateful for, that this project was even made possible, and for all the kind help and expertise we have been provided with. We would like to thank for the experimental data and for the arrangement of and help with experiments at Rigshospitalet.

A few other persons deserves to be mentioned for their contributions to the project.

Michael Kaare Rasmussen whose project is also a spin-off. We appreciate his effort with upgrading the structured light scanner and his participation in the experiments.

Professor Rasmus Larsen, who most kindly took the time to help us solve mathematical problems.

Finally we would like to thank the PET and Cyclotron Unit at Rigshospitalet for letting us perform the experiments there.

Abbreviations

BOC	Basin Of Convergence
DLP	Digital Light Processing
DTU	Technical University of Denmark
HRRT	High Resolution Research Tomograph
ICP	Iterative Closest Point
IMM	Institute for Informatics and Mathematical Modelling
IR	Infra Red
MRF	Markov Random Field
NDI	Northern Digital Inc.
PCA	Principal Component Analysis
PET	Positron Emission Tomography
RMS	Root Mean Square
SLS	Structured Light Scanner
SVD	Singular Value Decomposition

Contents

1 Introduction

1.1 Background and Objectives

High resolution medical imaging modalities are highly sensitive to patient movement during image acquisition. One example of patient movement during a dynamic Positron Emission Tomography (PET) recording is shown in figure 1.1. Different techniques are being used to overcome the issue – these vary from stabilizing fixtures to optical motion tracking systems, all of which require significant preparation and the patient's cooperation. Additionally they add to the perceived discomfort of undergoing the scanning procedure, which for the most part ranges from several minutes to a few hours.

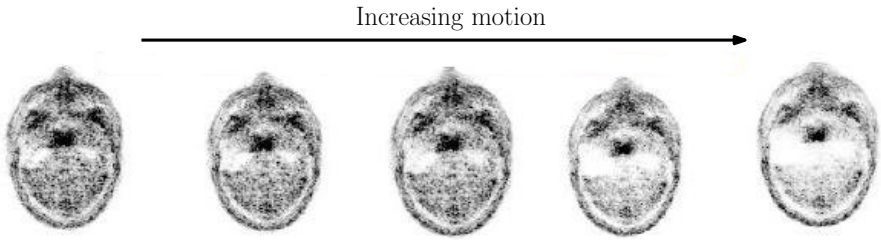


Figure 1.1: The effect of patient movement on a dynamic PET recording. Notice the white regions of no diagnostic value that appear with increased motion. Modified from [1].

Research is ongoing to implement a motion tracking system for the High Resolution Research Tomograph (HRRT). The HRRT is used in brain research, and the approach for tracking the movement of the brain is based on structured light 3-D surface registrations of the patient's face. The system consists of a structured light source, two video cameras and computer software. The results of surface registrations are three dimensional point clouds.

In order to do motion correction, the relative movement of the head must be calculated, for all six degrees of freedom, on every frame to some reference. In our work, most of these calculations are based on the iterative closest point (ICP) algorithm. ICP is a term that covers the use of an iterative algorithm that estimates the best alignment of two point clouds. The algorithm exists in many different variants and can be tailored to use different features of the point clouds depending on the circumstances.

The aim of this report is to investigate different approaches to registration in terms of speed and accuracy when used on partial surfaces of human faces situated in the HRRT. Objective evaluation also includes algorithm stability and computational complexity.

The report deliberately covers the following topics:

- First we describe the features of the structured light scanner. Challenges in regard to *in situ* recordings are briefly discussed. The section also introduces the necessary mathematical concepts regarding transformations in 3-D space and methods for working with point clouds.
- The ICP algorithm and its variants.
- Non linear alternatives for alignment error minimization.
- The equipment and methods we used for collecting experimental data, and the methods for preprocessing of data that we employed.
- Four sections presenting the results and evaluating *convergence*, *accuracy*, *robustness* and *computational times* of the algorithms.
- Conclusion are drawn based on our results.

1.2 The HRRT

Since the invention of Positron Emission Tomography in the 1970's [37], resolution has been ever increasing.

The HRRT is a dedicated brain scanner equipped with 120,000 scintillator crystals. The scintillators will register decay events of a radioactive tracer that has been injected into the subject on beforehand. The purpose of such an examination is to produce physiological 3-D images of the brain, showing the differences in metabolic activity throughout the brain. Both dynamic sequences and static images can be recorded.

The HRRT allows for a *full width half maximum* resolution of down to 1.4 mm [30]. This is much lower than the average head movement of patients in a PET scanning situation [8] – even when using a rigid head restraint [14].

Correction of head movement on high resolution imaging modalities is a field of active research. Most approaches are based on optical tracking of retroreflective

markers that are fixed to the patient using sticky tape, wet caps or bandaids [28].

For hardware based motion tracking, the NDI Polaris Vicra system is used at several PET centers [20]. It provides an accuracy of 0.35 mm RMS [39] within a pyramidal volume. Under ideal circumstances this system resolves the issue of motion blurriness to a great extend. However it quite complicated to operate.

The aim of the ongoing research carried out by Ph.D. student Oline V. Olesen at Rigshospitalet, DTU IMM and Siemens Healthcare, is to make the optical tool tracking system obsolete and to establish a reliable motion tracking routine that requires little user interaction. Such a system should potentially be built into PET scanners in the future [29].

1.3 Anatomical aspects

The success of the described motion tracking approach depends on the assumption that movements of the face are rigidly connected to movements of the brain.

It is easier to assume rigidity between the brain and the skull, since the brain is tightly encapsulated in the skull and dura and cushioned by cerebrospinal fluid. Although physical activity may cause some tension on the medulla, in a PET scanning situation this will have very little effect.

The face however is not bound rigidly to the brain at all. The attempt to track movement of the brain based on surface registrations of the face is a risky endeavor. Even in very cooperative subjects, facial expressions caused by eye movement, sneezing or affections cannot be prevented totally. Under the skin, the face contains many muscles, some of which can be seen in figure 1.2. While the eyes and the skin covering the orbit are connected very loosely, the skin on the nasal cartilage can hardly be moved.

These are strong arguments to base the motion correction mainly on the immobile regions of the face, namely the nose, the nasion and the forehead. It is however beyond the scope of this work to compare motion tracking approaches of different parts of the head. Nonetheless it is important to know what factors may contribute to bad results.

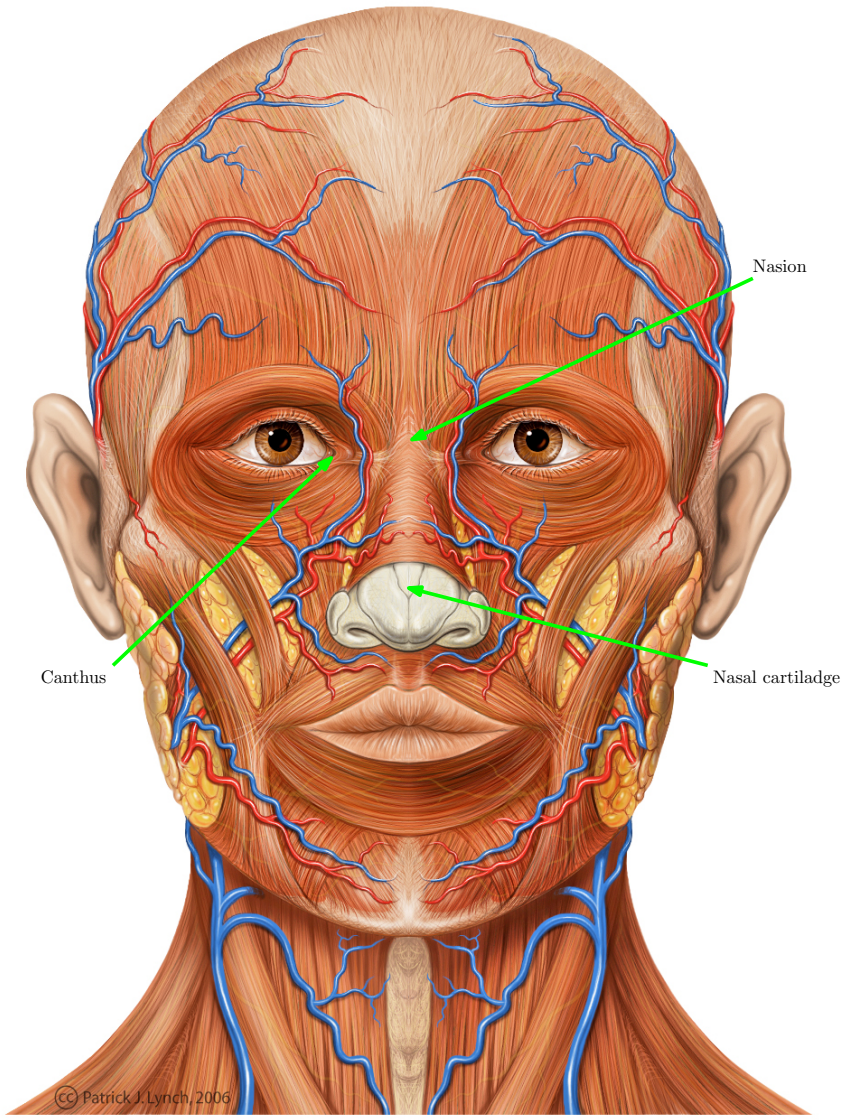


Figure 1.2: Frontal view of the superficial muscle layers of the face. Created under a creative commons license by Patrick J. Lynch.

2 Theory

This project revolves around point clouds. Mathematically they can be regarded as sets of points

$$\mathbb{P} = \{p_1, p_2, ...p_N\}\ , p \in \mathbb{R}^3$$

Although it is known that the point clouds resemble the surface of human faces, shape information is encoded indirectly. This means that for two individual point clouds correspondence between the points is not known. Any characteristic information has to estimated by proper mathematical treatment. Methods for doing so will be explained in this chapter. They serves as a foundation for understanding the ICP algorithm and its variants.

A brief explanation of how the point clouds are acquired will be given in section 2.1. Section 2.2 presents different methods for rigidly transforming point clouds in 3-D space. Finding neighboring points is in section 2.3 and estimation of normals and surface curvature in section 2.4 and 2.5 respectively. Finally section 2.6 presents a method that reconstructs surfaces from point clouds.

2.1 Structured light scanner

Acquisition of 3-D models can be done using the structured light technique. Compared to other scanning techniques, the required hardware for a structured light system can be fairly simple and cheap. Whereas other techniques scan one point at a time, the structured light system is able to capture the full field of view at once, which makes it ideal for the application presented in this project. The system captures images of an object and calculates a 3-D point cloud representation of the surface. For our work, the relevant points regarding the system require only an understanding of the basic concepts.

A structured light scanner basically consists of a projector, one or more cameras and a computer. For instance, the composition of one of the systems used for the HRRT is:

- A small projector that projects visual light.
- Two cameras mounted on either side of the projector on a frame.

These components are shown in figure 2.1.

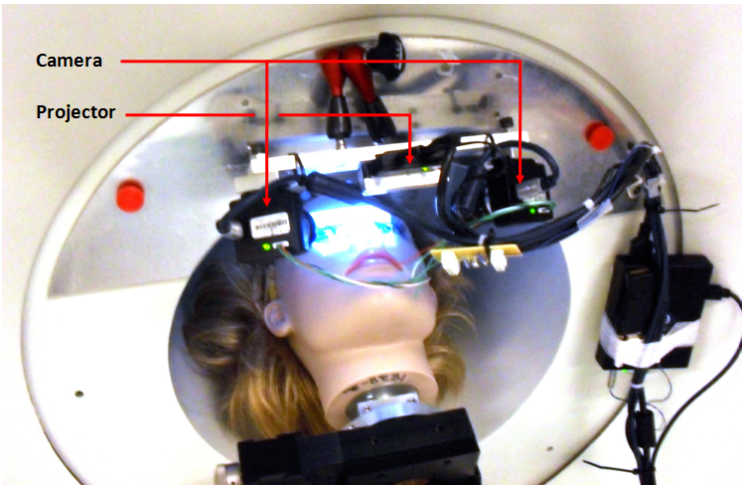


Figure 2.1: A structured light scanner system. An area of the face can be seen to be illuminated by the projector. The cameras are pointed at the illuminated area.

The projector is used to project defined light patterns onto the object of interest. When perceived from the cameras point of view, the pattern appears distorted due to change in perspective and the surface's curvature. This is shown in figure 2.2. The distorted light pattern is captured by the cameras. If an exact relationship between the cameras and projector is known, or established by calibration, it is possible to calculate points in 3-D representing the surface of the object [29].

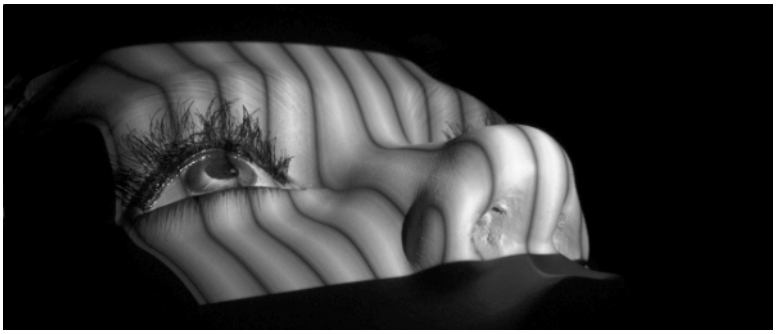


Figure 2.2: A regular grid light pattern is projected upon a mannequin face. The picture shows what the left camera from figure 2.1 sees. Notice how the regular pattern appears distorted.

Structured light systems have certain limitations due to the projection of the

light patterns.

- **Non-uniform point sampling density**. Consider a surface that runs almost perpendicular to the projection plane. The projected light pattern will be extremely distorted and only very few points can be calculated. Thus the sampling density varies as a function of the angle between projection plane and surface.

- **Pattern disrupts**. If the pattern cannot be seen by the cameras then no points can be found.

    - Shadows. A common problem where the surface of the object blocks the line of sight to the light pattern. This can be overcome with the use of multiple cameras or multiple recordings from different angles.

    - Reflective surfaces cannot be properly scanned.

    - Fine details, such as hair, have refractive properties and are usually to small to be resolved.

2.2 Representations of rotation in 3-D

Rotations in three dimensional space can be represented in different ways. Each representation has its advantages and disadvantages. In the following, representations using matrices, axis angle and quaternions will be explained.

2.2.1 Rotation matrices

In a standard Cartesian 3-D coordinate system three basis rotation matrices are defined. One to rotate around each of the three axes, as shown in figure 2.3.

For rotations of angles α, β and γ around the x, y and z axis respectively, the

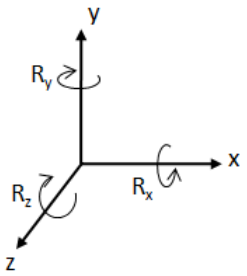


Figure 2.3: Rotation using matrices.

rotation matrices are

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

$$\mathbf{R}_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix}$$

$$\mathbf{R}_z(\gamma) = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

All rotation matrices are *orthogonal* with a determinant equal to one.

Rotation of a points is simply a matter of multiplying the rotation matrix from the right with a column vector containing the point coordinates. N points can be rotated via one matrix multiplication by storing them in a $3 \times N$ matrix and multiplying them on the rotation matrix from the right. In software packages such as Matlab, this is an efficient way of applying a rotation to many points at once. In our experiments, we therefore stored point sets in this way.

Since matrix multiplication is not commutative, the order of multiplication is important. To specify a rotation around the origin, in general three angles have to be given. We use the x, y, z convention, meaning that a rotation matrix can be factorized $\mathbf{R}_{\alpha\beta\gamma} = \mathbf{R}_z(\gamma)\mathbf{R}_y(\beta)\mathbf{R}_x(\alpha)$.

For small angles, $\cos\theta \approx 1$ and $\sin\theta \approx 0$. This lets one linearize the rotation matrix as follows

$$\mathbf{R}_{\alpha\beta\gamma} \approx \begin{bmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{bmatrix} \tag{2.1}$$

2.2.2 Axis Angle

Instead of representing a rotation by three angles around three fixed axes, the axis angle representation defines one axis given by a unit vector, $\vec{e}$, and one angle, θ, as shown in figure 2.4.

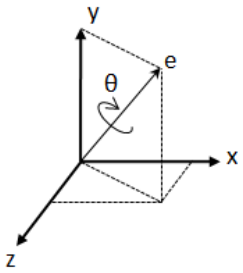


Figure 2.4: The axis angle representation for rotations in three dimensional space.

Notationally the axis angle representation may be written

$$\langle \vec{e}, \theta \rangle = \left(\begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix}, \theta \right)$$

This representation is simple and somewhat intuitive. It is good for comparisons, but calculations can be cumbersome. Convertion between rotation matrices and the axis angle representation can be done as shown in appendix E.3 and E.4. Therefore in our results, the axis angle representation will be used.

2.2.3 Quaternions

Quaternions take the form of four dimensional complex numbers. A quaternion is defined as

$$Q = q_w + q_x \mathrm{i} + q_y \mathrm{j} + q_z \mathrm{k}$$

where q_w is called the scalar part and the rest is called the vector or imaginary part. The elements (q_w, q_x, q_y, q_z) are real, and the imaginary units obey:

$$\mathrm{i}^2 = \mathrm{j}^2 = \mathrm{k}^2 = \mathrm{ijk} = -1$$

Conjugation and magnitude

As an extended complex number a quaternion has a conjugate and a magnitude defined as:

$$\begin{aligned} q &= w + x\mathrm{i} + y\mathrm{j} + z\mathrm{k} \\ q^* &= w - x\mathrm{i} - y\mathrm{j} - z\mathrm{k} \\ \|q\| &= \sqrt{qq^*} = \sqrt{w^2 + x^2 + y^2 + z^2} \end{aligned}$$

Inverse and unit quaternion

The inverse of a quaternion is an important property. It is defined as:

$$q^{-1} = \frac{q^*}{qq^*}$$

If the quaternion has a magnitude of one, it is called a unit quaternion:

$$\|q\| = 1 \;\Rightarrow\; q^{-1} = q^*$$

Multiplication

Quaternion multiplication is a very useful property. If a quaternion is represented by its scalar and vector part ($q_i = (q_{wi}, \vec{v_i})$), multiplication might be done using:

$$q_1 q_2 = (q_{w1} q_{w2} - \vec{v_1}\vec{v_2}, \vec{v_1} \times \vec{v_2} + q_{w1}\vec{v_2} + q_{w2}\vec{v_1})$$

The quaternion product can also be found using matrix multiplication:

$$q_1 q_2 = \begin{bmatrix} x_1 & y_1 & z_1 & w_1 \end{bmatrix} \begin{bmatrix} w_2 & -z_2 & y_2 & -x_2 \\ z_2 & w_2 & -x_2 & -y_2 \\ -y_2 & x_2 & w_2 & -z_2 \\ x_2 & y_2 & z_2 & w_2 \end{bmatrix}$$

As the quaternion product can be found using a matrix multiplication, it should come as no surprise that quaternion multiplication is *associative* but *non-commutative* meaning:

$$(q_1 q_2) q_3 = q_1 (q_2 q_3)$$
$$q_1 q_2 \neq q_2 q_1$$

Rotations

Unit quaternions can easily be converted to axis angle representations. The connection between the axis angle representation $(\langle e_x, e_y, e_z \rangle, \theta)$ and unit quaternion, q, is the following:

$$q = \cos\frac{\theta}{2} + e_x \sin\frac{\theta}{2}\mathrm{i} + e_y \sin\frac{\theta}{2}\mathrm{j} + e_z \sin\frac{\theta}{2}\mathrm{k}$$

Obviously, since the length has to be one, the scalar part is given by the vector part, meaning that the unit quaternion can represent three degrees of freedom.

Quaternions do not provide the same intuitive understanding of a rotation as axis and angle, but they can be more convenient to do calculations with, although they have their own set of arithmetic definitions. The advantages of using quaternions are a lower number of necessary calculations and the avoidance of singularities and ambiguities – e.g. *Gimbal lock*. Further, they make smooth interpolation between rotational states easy.

Quaternion multiplication is the operation used to rotate a point or vector. For instance to rotate the point, P, using the quaternion, q, one might proceed as follows.

The point is represented as a quaternion, p, where the scalar part equals zero:

$$p = (0, \vec{P})$$

The rotated point, p', is then be obtained by multiplying:

$$p' = qpq^{-1}$$

2.3 Nearest neighbor search

Several parts of this work require the computation of nearest neighbors. Given a set of N points, $\mathbb{P} = \{p_1, p_2, ...p_N\}$, and a point q, the problem consists in finding the point in $\mathbb{P}$ that is closest to q. This has practical uses in various applications, and therefore many different methods have been developed.

A basic idea is to calculate distances to all points in $\mathbb{P}$ and choosing the one with the shortest distance. This is called *brute-force search*, *exhaustive search* or the *naive approach*. The method is simple, but scales only linearly with the number N of points in $\mathbb{P}$. Expressed in *Big O notation*, this is $O(N)$.

If many nearest neighbor searches are performed within $\mathbb{P}$, it is advisable to represent $\mathbb{P}$ in a structure that allows for more efficient searches.

One possibility is to precompute which region in space are closest to a given point in $\mathbb{P}$, reducing subsequent searches to the question of which of those regions q is in. This is the approach followed by the *Delaunay triangulation* described in section 2.3.1.

Another method is to divide the points in $\mathbb{P}$ into groups according to their location. The nearest neighbor search is accelerated by conditionally excluding many points and thereby reducing the number of distance calculations. One such method is the *kD tree* [3], which will be detailed in section 2.3.2.

2.3.1 Delaunay triangulation

In a 2-D point set, the Delaunay triangulation is performed as follows. Three points form a valid triangle if the following condition is met

- The circumcircle of the triangle must not contain any other points from the point set (points are allowed to be on the rim of the circumcircle).

The Delaunay triangulation is not necessarily uniquely defined and might not be defined at all – e.g. for points lying on a straight line. A 2-D Delaunay triangulation is shown in figure 2.5a.

The *dual graph* of a 2-D Delaunay triangulation connects the centers of the circumscribing circles to a new diagram called the *Voronoi diagram* or *Voronoi tesselation*. For the Delaunay triangulation in figure 2.5a, the corresponding

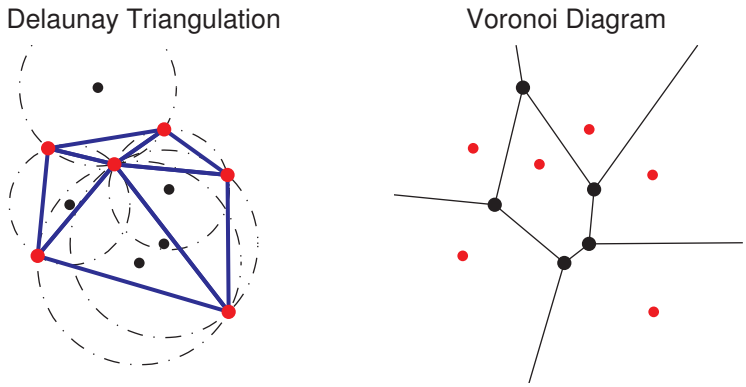


Figure 2.5: (a): A triangulation of six random points (red). The circumcircles and their centers are also shown (black). (b): The Voronoi cells and their corners (black) and the vertices of the Delaunay triangulation (red).

Voronoi tesselation is shown in figure 2.5b. Each point in $\mathbb{P}$ has a Voronoi cell, which is all the area for which the point is the closest.

These concepts can be expanded to higher dimensions, specifically 3-D. Here the Delaunay triangulation actually consists of tetrahedrons – each of which is defined by four points that are uniquely circumscribed by a sphere. Again the dual is the Voronoi tesselation, and this allows nearest neighbor searches by point location.

Several algorithms exist for the creation of both Delaunay triangulations and Voronoi tesselations. Since they are dual graphs, knowledge of the one can quite easily yield the other. Voronoi tesselations can be created using the *Bowyer-Watson algorithm* [6, 38] which can be implemented to scale no worse than $O(n^2)$ and typically $O(N \log_2 N)$ [33].

Point location

Voronoi tesselations contain the spatial information needed for nearest neighbor searches. It is however not trivial to determine which region a query point q is located inside. This poses the *point location problem* for which several solutions exist. A point location algorithm exists that can answer queries with complexity no worse than $O(\log_2 N)$ [23].

2.3.2 kD trees

In order to hold points in a kD tree, the set $\mathbb{P}$ is split by finding the median of all points' first coordinates. The one point corresponding to the median becomes the *root* of the tree. Next, the two resulting subsets are split based on the median of their second coordinates. The process – known as *binary space paritioning* – generates a balanced binary tree containing all points of $\mathbb{P}$.

Figure 2.6 illustrates the creation of such a kD tree for points in two dimensions. Though difficult to illustrate the process can easily be thought extended into higher dimensional spaces. In $\mathbb{R}^3$, which is of interest in this report, point clouds can be subsequently divided into two equally sized sets by planes that are orthogonal to the x, y and z axis in turn.

The computational complexity of creating a kD tree is no higher than $O(kN \log_2 N)$ [11]. This corresponds to sorting the point with regard to every of their coordinates, – that is k times a $O(N \log_2 N)$ sorting operation.

By using kD trees the computation time for finding nearest neighbors in Euclidean metrics can be greatly reduced. Assume that a kD tree has been constructed containing the points of set $\mathbb{P}$. Locating the nearest neighbor to q within $\mathbb{P}$ can be done in the following algorithmic way.

1. Move down the tree starting at the root comparing coordinates according to the actual splitting dimension until a leaf node is reached.

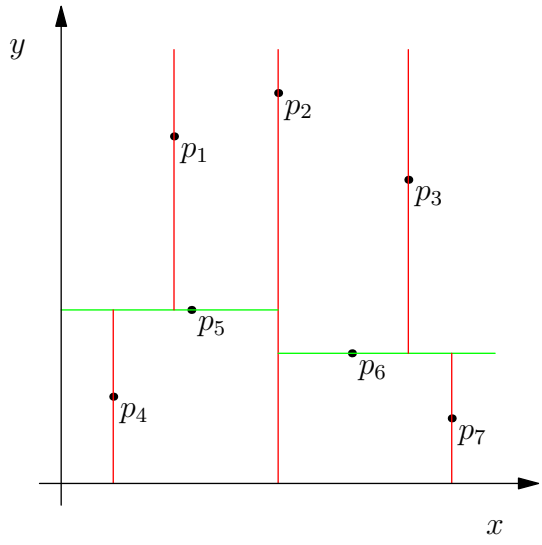
2. Mark the point at the located leaf node as *current best*, and calculate the distance d between q and current best.

3. Move up one level in the kD tree and determine the distance from q to this node. If the distance is shorter than d, define this node as current best and the distance to it as d.

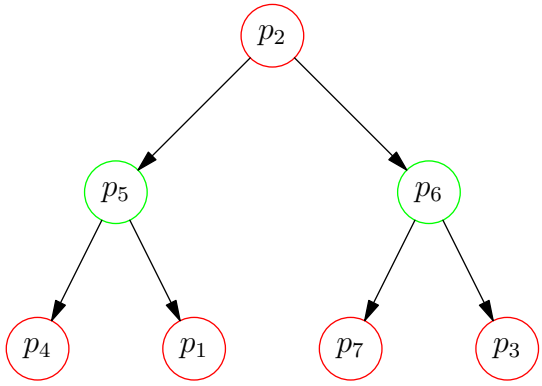
   If the distance from q to the current nodes' splitting plane is longer than d, exclude this nodes' other side branch and continue moving upwards until the root is reached.

   Otherwise the nodes' other side branch is searched through just like the whole tree.

4. When the top node is reached and all necessary side branches have been searched, choose the shortest of all candidates from the main search and eventual sub branch searches.



(a)



(b)

Figure 2.6: (a): A two dimensional point cloud which has been divided using *binary space partitioning*. (b): The correspoding kD-tree.

Depending on the sparsity of $\mathbb{P}$, very large or only smaller parts of $\mathbb{P}$ can be excluded from the nearest neighbor search. The reason is that if the distance to the first encountered leaf node is shorter than the distance to its parents splitting planes, then we are at the right node, and all other branches can be excluded. In this case, only a single distance calculation was done, and the complexity is $O(1)$. The expected complexity of the search algorithm is $O(\log_2 N)$ [11]. In the worst case, no point can be excluded from the search, and the algorithm will perform like the naive approach – that is $O(N)$.

Two simple modifications to the kD tree algorithm should not be unmentioned.

In performing *k-nearest neighbor* searches, the location algorithm will keep track of not only the nearest neighbor but a given number of nearest neighbors. We avoid to denote this number of neighbors k, since this variable is also used to denote the number of dimensions of the search space. Friedman et al. show that this modification does not add to the complexity of the algorithm [11].

An *approximate nearest neighbor* search aims to find good candidates for nearest neighbors while maintaining very low complexity. In kD trees, this might be implemented as a simplification of the location algorithm. It will traverse down the tree and return the point inside the leaf node as the approximate nearest neighbor. Obviously, this gives us the aforementioned $O(1)$.

2.3.3 Search performance

The discussion above suggests that the time consumed by nearest neighbor searches is very dependent on the algorithm employed. Furthermore, the implementation is critical. Lastly, in the case of kD trees, search performance will depend on the search space.

The creation of kD trees and Delaunay triangulations takes considerable time, while basically no preprocessing is necessary for a brute force search. Figure 2.7 shows the time consumed for creation of search structures as a function of the number of 3-D search points, N. In all cases precompiled Matlab 2010a functions were used on a modern Linux computer.

We demonstrate the influence of the geometrical relation between q and $\mathbb{P}$ by measuring the time needed to perform nearest neighbor searches in two situations. In situation 1 the query point is located inside the same $1 \times 1 \times 1$ bounding box as N uniformly scattered points. In situation 2, the query point is translated by one unit, such that is falls out of the bounding box. The situations are depicted in figure 2.8.

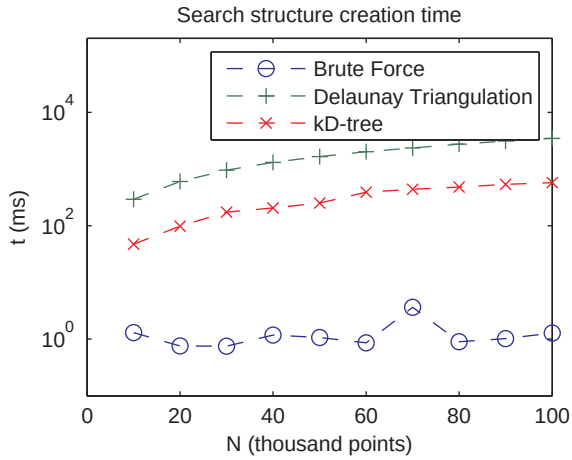


Figure 2.7: Time consumed for the creation of appropriate search structures in Matlab 2010a. The search structures were created using the functions `ExhaustiveSearcher()`, `DelaunayTri()` and `KDTreeSearcher()` respectively.

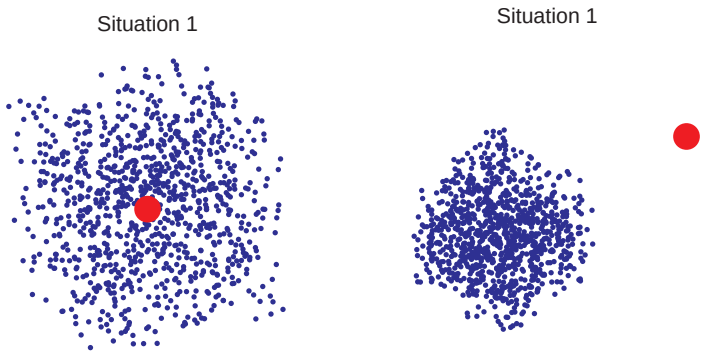


Figure 2.8: The two search situations. Search points are blue, while a single query point is shown in red. The search points are distributed uniformly within a cubic box. In situation 1 the query point is located within the box, while it is located outside in situation 2.

Figure 2.9 shows averaged search times for 1000 queries as a function of N. The standard Matlab functions were used. Clearly the brute force search performs worst, while kD trees are by far the fastest search method in situation 1. In situation 2 however, the Delaunay triangulation approach outperforms kD trees. It is seen that the location of the query points has no particular effect on the search performance of the Delaunay triangulation, while the kD tree approach is very sensitive towards it.

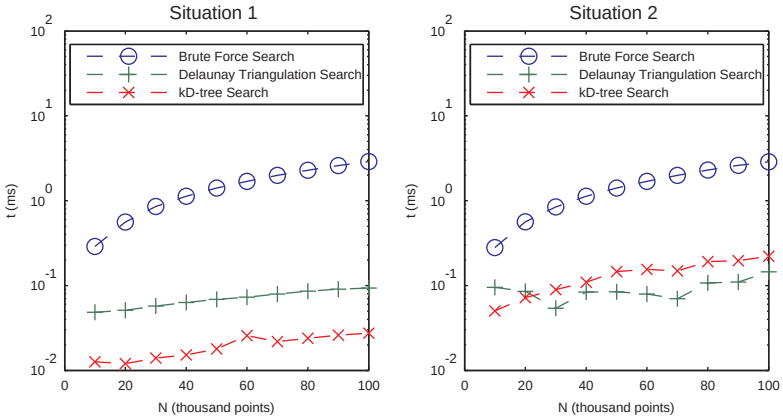


Figure 2.9: Nearest neighbor search times for the two situation depicted in figure 2.8. The times are averaged for 1000 over searches.

It is clearly seen that the creation of Delaunay triangulations and kD trees takes a considerable amount of time. When many neighbor searches are performed, the overall search time can however be reduced dramatically. The choice of search structure for 3-D nearest neighbor searches should depend on how many searches are done with the same query points, and on spatial relation between query and search points.

2.4 Principal component analysis

Principal component analysis (PCA) is used to describe variance in higher dimensional data.

Considering a number of scalar values, $\mathbb{S} = \{s_1, s_2, s_3, ...s_N\}$, $s \in \mathbb{R}$ with mean

$\bar{s}$. The variance is found by

$$\mathrm{Var}(\mathbb{S}) = \frac{1}{N}\sum_{i=1}^{N}(s_i - \bar{s})^2 \tag{2.2}$$

It is understood that the variance is along the data's first – and only – dimension. For data of higher dimensions, e.g. a set of three dimensional points, $\mathbb{Q} = \{q_1, q_2, q_3, ...q_N\}$, $q \in \mathbb{R}^3$ with centroid $\bar{q}$, the 3×3 *covariance matrix* is given by

$$\mathbf{C}(\mathbb{Q}) = \frac{1}{N}\sum_{i=1}^{N}(q_i - \bar{q})(q_i - \bar{q})^{\mathrm{T}}$$

though this is true for any dimension, and it is easily seen that this expression simplifies to (2.2) for one-dimensional data.

The covariance matrix holds information not only about how much the data points deviate from the centroid but also in which direction if there is any correlation. Specifically, the eigenvectors of the covariance matrix are called the *principal components* and the eigenvalues associated to them quantize the variance in their direction. They form an orthogonal basis and are named first, second and third principal component with descending eigenvalues. The first principal component points in the direction of most variability and the second principal component points in the direction of most remaining variability as seen in figure 2.10.

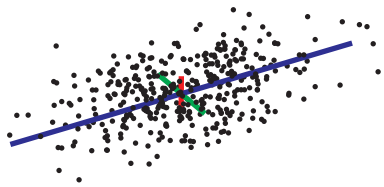


Figure 2.10: Principal component analysis of a three dimensional point cloud. The blue line represents the first principal component along which the points have the greatest variability. The second and third principal components are the green and the red lines respectively.

2.4.1 Estimation of normals

For point cloud data, surface normals can be estimated for every point using a PCA of neighboring points. In dense point clouds resembling surfaces, neighboring points can be assumed to lie more or less in a plane. The first two principal

components will span this tangent plane, while the third principal component will point in the direction normal to this plane. It can thus be used as a normal vector estimate [17].

The estimate will depend upon the point neighborhood. One strategy is to use a fixed number of neighbors. Another is to use all neighbors that are within a given range of the point in focus. The computational complexity for both methods is high and they scale only linearly to the number of points. However both methods can be realized using kD trees.

For every tangent plane, two opposite pointing normals are defined, and for some applications, normals should be oriented – e.g. the angle between normals in a close neighborhood should not exceed $\pi/2$ rad. Using the PCA method outlined above, normals will usually lie on both sides of the surface of interest. To address the issue on 3-D data a *normal flipping* scheme has been suggested [17].

For 2.5-D data, the problem is somewhat simpler. In knowing that the underlying surface can be described as $z = f(x, y)$, normals can be flipped to satisfy either of the inequalities $\vec{n} \cdot \langle 0, 0, \pm 1 \rangle > 0$.

In triangulated point clouds, for every point, the *1-ring neighborhood* consist of all vertices that are corners in the triangles that the investigated point is a corner in, as is seen in figure 2.11. The faces of the triangles are flat, and have normals which can be computed easily. In such point clouds, normals can be determined by averaging the normals of triangles that a given vertex is a corner in. This reduces computational complexity a lot since ordered neighborhood information is available. However the issue of normal orientation remains.

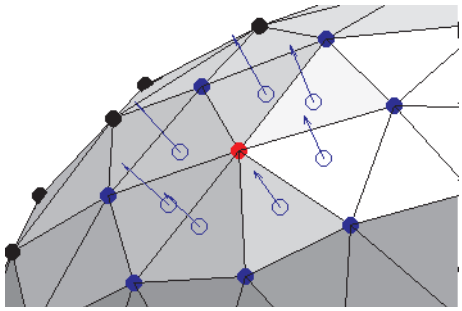


Figure 2.11: View on a triangulated point cloud. The point under investigation is shown in red. The 1-ring neighborhood is the blue points. Determining face normals is a simple matter of geometry. Point normals can then be estimated. In this case a normal for the red point is found as the average of the seven blue vectors.

Data stemming from a single camera in a structured light scanner are 2.5-D data in nature. They can be equipped with oriented normals using PCA at a considerable amount of computation. Most of our data will be preprocessed and triangulated. In these cases, normals will naturally be computed using the triangulation.

2.5 Curvature

Curvature is a measure of concavity or convexity. In the analytical sense and for a natural parameterized curve, it is defined as the norm of the second derivative of the parameterization. Graphically this is the change in tangent slope as one moves along the curve at a constant speed. Considering a parameterized circle

$$c(t) = \begin{bmatrix} R\cos t \\ R\sin t \end{bmatrix}$$

the arc length is given by $s = Rt$ with inverse $t = \frac{s}{R}$, so the natural parameterization is given by

$$c(s) = \begin{bmatrix} R\cos \frac{s}{R} \\ R\sin \frac{s}{R} \end{bmatrix}$$

The norm of the second derivative gives the curvature $\kappa(s)$ of the circle

$$\kappa(s) = \|c''(s)\| = \left\| \begin{bmatrix} -\frac{1}{R}\cos \frac{s}{R} \\ -\frac{1}{R}\sin \frac{s}{R} \end{bmatrix} \right\| = \frac{1}{R}$$

It can been see that curvature is given by the inverse of the radius of the circle. This means, that curvature can be thought of as the inverse radius of an osculating circle that locally approximates the curve, as seen in figure 2.12.

2.5.1 Surface curvature

For surfaces, the definition of curvature is not straight forward. Noting that a surface may bend differently in different directions, a single parameter might not cover all there is to it. Using the surfaces tangent plane at a given point, one might intersect the surface with planes perpendicular to the tangent, including that specific point. This gives two dimensional curves that can be analyzed using the method above. This is shown conceptually in figure 2.13. However, an infinite number of such intersections can be made. Choosing the two that yield the highest and the lowest curvature values, one gets the *principal curvatures*

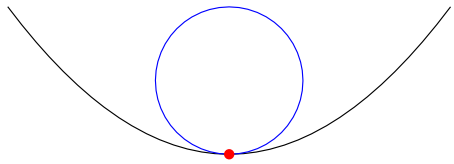


Figure 2.12: The curvature at the vertex of the shown parabola equals the inverse of the osculating circle shown in blue.

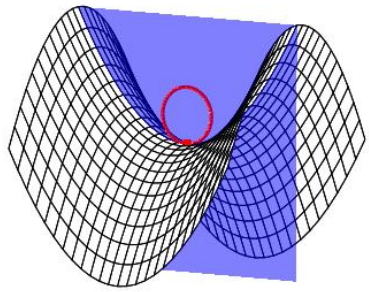


Figure 2.13: A surface is intersected by a plane that is perpendicular to the tangent plane at the red point. The resulting curve has an osculating circle inside the plane.

of a surface. It is worth noting that the two corresponding intersecting planes are mutually perpendicular.

To arrive at a scalar valued function for surface curvature, principal curvatures are averaged to yield *mean curvature*, or their product is taken, giving *Gaussian curvature*.

2.5.2 Point cloud surface curvature

For point clouds, the curvature of the underlying surface can only be estimated and several methods have been proposed. One strategy is to locally fit an analytical surface to the points and deriving the curvature from the derived coefficients, but this is computationally very complex, and will not be considered here. Other methods are based on discrete schemes that try to approximate curvature as closely as possible. Pauly, Gross and Kobbelt [32] calculate the eigenvalues of the covariance matrix of a points neighbors, λ_0, λ_1 and λ_2 where $\lambda_0 \leq \lambda_1 \leq \lambda_2$. They then define the ratio

$$\sigma_n = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{2.3}$$

calling it the surface variation based on n neighbors. They demonstrate, that surface variation is a good indication of the the surface mean curvature. Points lying in a plane will have $\sigma_n = 0$ and for isotropically distributed points the curvature is $\sigma_n = 1/3$. The results of Paulys method are seen in figure 2.14 (a).

Gumhold, Wang and MacLeod [15] take a points average distance to its neighbors, μ, and the same points distance to its neighborhood centroid, d, and estimate surface curvature as

$$\kappa = \frac{2d}{\mu^2}$$

The result of applying this curvature measure is shown in figure 2.14 (b).

Miao, Feng and Peng [26] take an geometrically interesting approach. From every point they grow a sphere with radius r on the points normal minimizing the squared distances from neighbors to the sphere. Denoting a point p_0 with normal n and its neighbors $p_1, p_2, ...$, the objective function for squared sphere neighbor distances becomes

$$e(r) = \sum_{i \in 1,2,...} (\text{dist}(p_0 - r\vec{n}, p_i) - r)^2$$

They further enhance this by multiplying every term with a standard Gaussian weighting function. This makes neighbors far from p_0 have less weight in the

objective function. Minimization of the objective function is done using non linear methods, and the surface curvature estimate is the reciprocal of the found argument r. Figure 2.14 (c) shows this method applied to a point cloud.

Gaussian curvature in the analytical sense is defined as the product of a surfaces principal curvatures. A popular discretized scheme takes offset in a point clouds triangulation representation. It uses the sum of the angles of the triangle corners that meet at a given point, θ_i, and the sum of the areas of those triangles areas, A_i in the formula

$$G = \frac{2\pi - \sum_i \theta_i}{\frac{1}{3}\sum_i \mathrm{A}_i} \tag{2.4}$$

For this method to be applicable, the point cloud in question needs to be triangulated. There are problems with this approach, namely at points that are on the edges of a triangulation, which do have fewer neighbors in their *1-ring* neighborhood. These points would require special treatment for a useable estimate to be made. The Gaussian curvature estimate for a triangulated point cloud is seen in figure 2.14 (d). It clearly shows the aforementioned issue.

The preceding discussion and the results seen in figure 2.14 lead to the conclusion, that the most usefull curvature estimate for our purposes is Paulys. It provides consistent results and is easily implemented. In addition, computational complexity is lower than for the other methods.

2.6 Surface Reconstruction

The ability to describe the surface of an object as a triangulated mesh with normals is of major importance in applications such as computer graphics. Methods for calculating normals and triangulations and performing surface reconstruction from point clouds is therefore a well studied problem, and several methods have been developed over the years. Their focus has been on improvements with regard to:

- Optimal meshes: Avoid huge and oddly shaped triangles. Triangles should be as close to equilateral as possible.
- Outliers: Outlying points have a tendency to create odd triangles, and should therefore be removed.
- Holes: Small holes and gaps in the point cloud should be closed.
- Consistent normals: All normals should point in the same direction with respect to the surface.

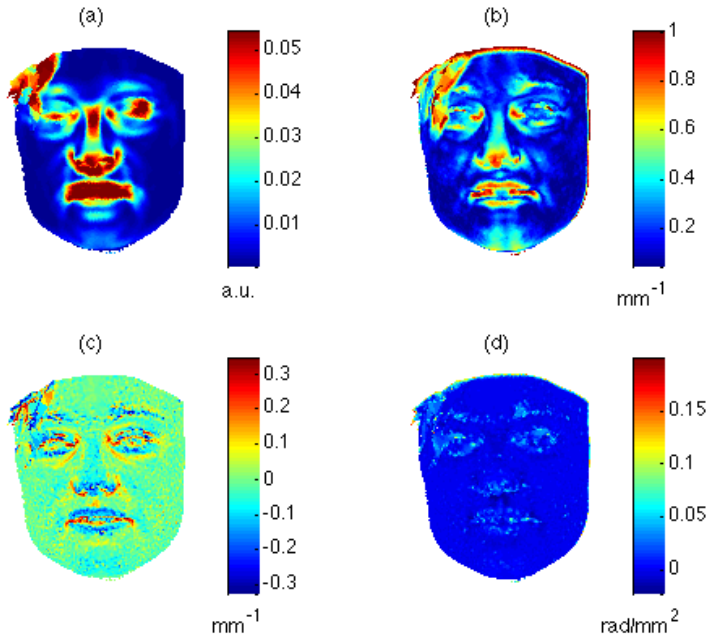


Figure 2.14: A point cloud of app. 20 k points with color coding according to four different methods of curvature estimation. (a): Paulys method as given by equation 2.3 based on 200 nearest neighbors. (b): Gumhold curvature estimate [15] based on 200 nearest neighbors. (c): Miao et al. surface estimate using a Gaussian weighting function and a neighborhood of 50 points. (d): Gaussian curvature using *MRF surface reconstruction* of the point cloud.

- Computation times.

The current methods can largely be grouped into direct and implicit algorithms.

Direct methods seek to connect the existing points to form triangles – e.g. the *ball pivoting algorithm* which lets an imaginary sphere roll over the point cloud and connect points that the sphere can rest on.

Implicit methods create new vertices on the basis of a three dimensional *distance field* function $d(x, y, z)$. The space surrounding the point cloud is discretized into voxels, and every voxel is assigned a value that indicates its distance to the desired surface. The isosurface $d(x, y, z) = 0$ is then extracted. Finally, the surface might be remeshed to yield triangles that better fulfill the aforementioned criteria.

The method used in this report is implicit. It is called *Markov Random Field Surface Reconstruction* [31] and it is implemented in a free software program called MRFSurface. When the algorithm is applied to a point cloud, outliers are removed and new points are created in order to achieve a more even point density and to fill holes.

An example of the MRF surface reconstruction is shown on figure 2.15 and 2.16 where a part of a mannequin head can be seen before and after the reconstruction.

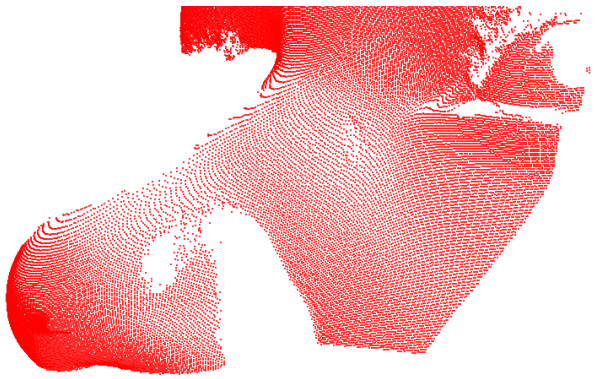


Figure 2.15: Point representation of the partial left side of a mannequin head. The left eye can be seen in the top right part of the picture and the nose in the lower left.

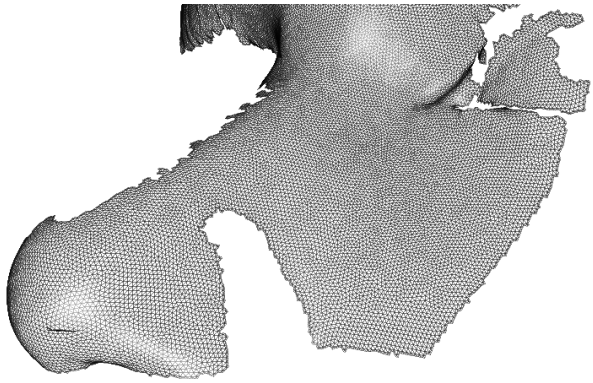


Figure 2.16: The surface reconstructed mannequin head. Compare to figure 2.15 and notice how outlying points around the eye have been removed, and how the small hole in the nose have been closed.

3 Point cloud registration

The optimal registration of point clouds is a substantial problem to which many authors have contributed. The overall goal is to apply a transformation to one of the clouds and bring it as close to the other as possible. We will call the first set of points model points and the other set data points. The convention is to apply the transformation to the data points in order to bring these to the best alignment with the model points. In this work, the model point set will be denoted $\mathbb{Q} = \{q_1, q_2, q_3, ...q_N\}$ and the data point set $\mathbb{P} = \{p_1, p_2, p_3, ...p_N\}$. In order to formulate the goodness of fit mathematically, an error metric, or objective function E is defined. A popular metric is the sum of squared errors – that is the squared distances from points in one cloud to their nearest neighbors in the other *after* appliance of the transformation τ

$$E = \sum_{i=1}^{N} \|\tau(p_i) - q_i\|^2$$

Transformations can be rigid or non-rigid, and the proper should be chosen according to the task and reasonable underlying assumptions. For head tracking purposes, the assumption of rigidity might be weak. But for practical reasons and since we're really interested in movements of the brain it is a reasonable assumption. In 3-D space, such a transformation has six degrees of freedom – three rotations and three translations – and the objective function becomes a function of six variables. Expressing a rotation in terms of the rotation matrix $\mathbf{R}$ and a translation in the vector $\vec{T}$ the problem may now be written

$$E = \sum_{i=1}^{N} \left\| \mathbf{R}p_i + \vec{T} - q_i \right\|^2$$

In data from structured light surface registrations however, point correspondence is not given. This means, we do not know which point q_i will be the nearest neighbor to p_i after transformation. This information can be approximated based on the nearest neighbor *before* transformation. The iterative closest point algorithm does this exactly. The non linear minimization methods presented later, cleverly choose some transformation and determine how it should be changed to decrease E.

3.1 The ICP algorithm

In order to address the issue of point correspondence, the ICP algorithm iteratively performs the following steps

1. Matching: for every data point the nearest neighbor in the model point set is found.
2. Minimization: the error metric is minimized.
3. Transformation: data points are transformed using the minimization result.

The algorithm is terminated based on the number of iterations or the relative change in the error metric. In many cases, the algorithm will converge quite rapidly, however several problems may arise:

- Multiple local minima in the error metric: the algorithm may converge towards one of the local minima instead of the global minimum.
- Noise and outliers: these cause the error metric to never be zero. Outliers may cause faulty results, especially at quadratic error weighting.
- Partial overlap: the point clouds may not resemble the same parts of an object. Partial overlap must be required though.

The ICP algorithm was introduced in 1991 by Chen and Medioni [7] and independently be Besl and McKay [4] and it was further developed by various researchers.

3.2 ICP taxonomy

Many researchers have proposed solutions to address the issues of the ICP algorithm. This has led to several different variants of the ICP algorithm. A taxonomy of ICP variants was suggested by [34]. It identifies six distinct stages of the algorithm:

1. Selection

2. Matching
3. Weighting
4. Rejecting
5. Error metrics
6. Minimization of error metric

The following sections will give a brief overview of every one of these and the strategies will be discussed according to the requirements for our implementation.

3.2.1 Selection

It may be beneficial to consider only some of the model and data points before applying the ICP algorithm. For instance outliers may be filtered based on some threshold. Or to reduce computational complexity, the amount of points may be reduced by random or uniform subsampling. This will speed up computations – especially the matching step.

The idea behind random sampling is to sample differently in every iteration of the algorithm in order to prevent any bias towards outliers [25].

Flat regions in the point clouds might contain redundant information of which some can be discarded. If the underlying objects color, curvature or tangent normals are available, a strategy is to have these features represented in some predefined distribution.

For the point cloud data under investigation, no color information is available, but curvature and tangent normals can be estimated using techniques presented in previous sections. Thus, the aforementioned strategy might be used to raise sampling density in feature-rich regions such as the nose tip and eye brows. This might not only reduce complexity but also increase convergence.

To achieve a good distribution of surface normals among the sampled points, these should be put into buckets of similar normal direction. In this work, buckets were based on the angles to the basis vectors. For a normal of unit length

$$\vec{n} = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}$$

the angles to the x, y and z-axis are calculated simply as

$$\operatorname{acos}(n_x) \quad \operatorname{acos}(n_y) \quad \operatorname{acos}(n_z)$$

which are simple and fast to calculate. A point cloud that was bucketed in this way is shown in figure 3.1. Subsampling then happens across these buckets, subsequently picking points from each bucket until enough points have been collected.

Figure 3.1: Point cloud of a human face. Points of similar normal direction are grouped in color coded buckets. Here the normal-space was divided into 27 regions.

In the case of curvature based sampling, points are simply put into buckets based on their curvature values, and sampling is done across buckets.

3.2.2 Matching

Matching accounts for the pairing of points from the data point cloud to the model point cloud. Finding the nearest neighbors is usually the most computational intensive step in the ICP algorithm. The *brute force* method simply calculates distances to all neighbor candidates and picks the closest. This method scales linearly, and is only competitive for few lookups or higher dimensional data.

Techniques might be employed to speed up nearest neighbor searching. These include kD trees and Delaunay triangulations, both of which provide greatly enhanced lookup times at the expense of some preprocessing. In most iterations of the ICP algorithm, data and model point clouds will be quite close to each other. We therefore favor kD trees and use them to speed up the matching step in our implementation.

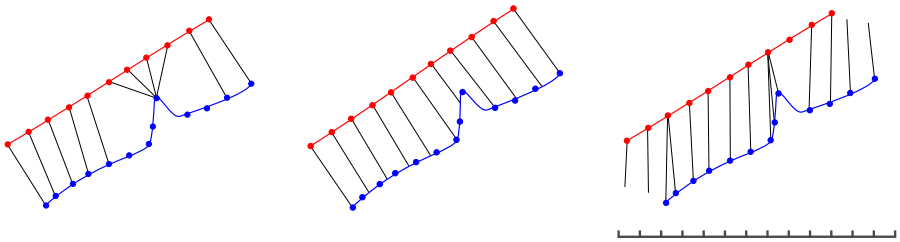
Alternatively, point matching can be done by finding the *line-surface intersection* [7] – a technique commonly referred to as *normal shooting* [34] and *ray shooting* in the computer graphics scene. Letting ℓ define the line originating in data point p with direction of normal $\vec{n}_p$

$$\ell : \begin{bmatrix} x \\ y \\ z \end{bmatrix} = p + t \cdot \vec{n}_p \qquad t \in \mathbb{R}$$

Every data point p is paired with the intersection of ℓ with the surface of the model point cloud. The method requires normals to be estimated in both model and data. Further, the model point cloud needs to be handled like a surface in some way. Generally speaking, *normal shooting* will requires a good initial alignment in order to converge. It might however be more resistant to noise in the data as shown conceptually in figure 3.2.

An implementation of *normal shooting*, for parametric 2.5-D surfaces of the kind $f(x, y) = z$, was described by Chen and Medioni [7]. We use surface reconstruction and perform ray shooting using a ray triangle intersection test given by Möller et al. [27]. If for a particular point, the line ℓ does not intersect any triangle, the point is not taken into account. This approach scales linearly with the number of points in the model point cloud – it serves only as a method of comparing the choice of matching strategy. Much more efficient algorithms exist [12], in particular for triangulated point clouds, since they form the basis for many tasks in computer graphics. Efficient algorithms for the determination of ray/point-cloud intersections also exist [35].

An even simpler and potentially more effective way of establishing point correspondence is the projection of data point onto to camera plane of the SLS. In calibrating the system, a relation between the 2-D camera plane and the 3-D coordinate space is established. The calibration takes into account lens distortion and the focal length along other parameters. Olesen gives the explanation of this relation in [29]. The idea of *reverse calibration* [5] is to project the 3-D coordinates onto the camera plane and match points in this discrete representation. This is shown for a 2-D example in figure 3.2. This method can be assumed to give worse matches in many cases, however it can lead to greatly reduced computation times overall [34]. In our experiments, we preprocess data, so that a reverse calibration as described is not possible.



Nearest neighbors Normals shooting Reverse calibration

Figure 3.2: ICP matching strategies in two dimensions. Blue points represent the model point clouds to which red data points are matched. In the reverse calibration case, the points are projected onto the discretized, lower dimensional camera view space.

The point clouds originating from the SLS contain about 20 k scattered points. Efficient matching of all these points cannot be done using the brute force method, so one of the discussed speed up methods should be employed.

3.2.3 Weighting

Matched point pair may be weighted differently based on their compatibility in some sense. Practically this means multiplying ever term in the error metric with a specific factor w. The factor could be based on distance, color, curvature or tangent normal directions.

Color information is not available, but tangent normals and curvature can be estimated, so these can be implemented for the structured light point clouds.

Normal compatibility weighting for a point pair with normals $\vec{n}_p$ and $\vec{n}_q$ can be done using the weight [34]

$$w = \vec{n}_p \cdot \vec{n}_q \tag{3.1}$$

For points with curvature values c_p and c_q in the range $[0; 1]$, we propose a Gaussian weight

$$w = \mathrm{e}^{-(c_p - c_q)^2} \tag{3.2}$$

To weight the point pair (p, q) according to their distance, Godin suggested [13]

$$w = 1 - \frac{\mathrm{dist}\,(p, q)}{\mathrm{dist}_{\max}} \tag{3.3}$$

in which dist_{max} is the maximum distance of all point pairs.

3.2.4 Rejection

Point pairs may be rejected after the matching step. This can be done on a statistical evaluation of the nearest neighbor distances – e.g. the 10 % worst point pairs may be rejected.

For point clouds that have only partial overlap, point pairs involving edge vertices might be rejected. This requires at least one of the point clouds to be triangulated. Two points are both edge vertices if the pair only appears in one triangle. If the point pair is present in two triangles, then it can only be concluded that one of them is not an edge vertex. This is illustrated in figure 3.3.

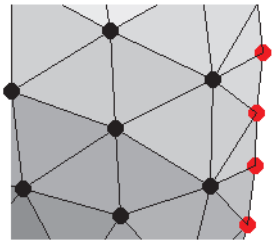


Figure 3.3: Part of a triangulated surface. Edge vertices are shown in red. The topological characteristic of these points is that the edges between two of them are part of only one triangle.

The point clouds under investigation are expected to have only partial overlap since the imaging region of the structured light scanners is a part of the face that changes as the patient moves. Both statistical and edge vertex rejection can be considered to overcome faulty alignments. A surface triangulation might be computationally expensive, and using a single triangulated model data set for many subsequent registrations can reduce this overhead.

3.2.5 Error metrics

The error metric defines the objective function that is minimized in every iteration of the algorithm. Two metrics are commonly used:

Point to point minimization [4] sums the squared distances of data points to model points. It might be expressed

$$E = \sum_{i=1}^{N} \left\| \mathbf{R}p_i + \vec{T} - q_i \right\|^2$$

In physical terms this metric can be visualized as the total potential energy of springs attached between matched point pairs.

Point to plane minimization sums the distances of data points to the tangent planes in which the matched model points reside. Its' mathematical formulation is

$$E = \sum_{i=1}^{N} \left[\left(\mathbf{R}p_i + \vec{T} - q_i \right) \cdot \vec{n}_i \right]^2$$

where $\vec{n}_i$ denotes the estimated tangent normals at the ith model point. The physical interpretation might be the total potential energy of springs that are free to move on model point tangent planes and fixed to the data points. The point to plane minimization lets flat regions slide along each other which can be advantageous.

Other error metrics can be imagined. For instance the L_1 norm could be used instead of the Euclidean norm above.

3.2.6 Minimization

A closed form solution exists for the minimization of the point to point error metric [2] and can be found in appendix C. It is based on the *singular value decomposition* described in appendix A. A closed form solution also exists for the equivalent quaternion formulation [9, 18].

The point to plane error metric only has a closed form solution after linearization of the rotation matrix $\mathbf{R}$, which is shown in appendix D. The linearization introduces an error, but in late iterations, rotations are expected to be small, and the linearized rotation matrix is close to the real one. The minimization was proposed and derived by [7].

For other metrics in general a closed form solution cannot be expected to exist. In that case non linear methods can be used. Using non linear methods is a delicate balance of acceptable accuracy and computational expense, it gives however the freedom to model the objective function as one likes.

3.3 Non linear methods

The ICP algorithm solves the correspondence problem by iteratively moving points to their current nearest neighbor. There is however another way to approach the problem. The objective function can be modeled to establish point correspondence dynamically, and it can then be minimized using a non linear optimizer.

For any combination of the six degrees of freedom, an evaluation of the objective function requires

- Transformation of the data point cloud
- Finding nearest neighbors
- Evaluation of the error metric

Metaphorically speaking and reduced to two variables, one can imagine the objective function as a hilly landscape. The goal is to find the lowest valley. The strategy is to move downhill as far as possible and to change direction if at some point the ground ascends. This will lead to a local minimum at some point.

The principle of moving downhill corresponds to moving along the negative gradient. In order to reduce the number of steps taken, one may take greater jumps based on the general direction outlined by previous path. The point to point and point to plane error metrics let us not derive an expression of their gradient, and thus it has to be approximated by finite differences. A stopping criterion might be defined as for the ICP algorithm – based on the relative change in the error or on iteration count.

Several non linear minimization algorithms exist, and they differ in robustness and speed. A popular choice is given by the *Levenberg-Marquardt algorithm*. It has been shown, that the application of this algorithm to the point cloud registration problem is competitive to the ICP algorithm [10] – both in terms of speed and robustness.

The strength of a non linear methods is, that the objective function can be modified in many ways. For instance it can be enhanced with weighting functions. Further the algorithm has been implemented various times in efficient versions.

3.4 An accelerated ICP algorithm

The demand for a fast converging point cloud registration algorithm has led to the combination of the ICP algorithm and concepts of non linear minimization.

Noting that the point to point error metric minimizes the mean square of errors, such that throughout the algorithm the error decreases monotonically [4], Besl and McKay proposed to extrapolate intermediate results during the ICP iteration sequence [4].

The optimization space for a rigid transformation is six dimensional. Using unit quaternions, one avoids the problem of singularities and ambiguities, and the transformation may be represented by a seven dimensional vector with the aforementioned six degrees of freedom:

$$\vec{q} = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \\ T_x \\ T_y \\ T_z \end{bmatrix}$$

The progress of the ICP algorithm is now considered a journey in the transformation space. Letting $\vec{q}_k$ denote the total transformation state vector at the kth iteration, one may define the change in every iteration as

$$\Delta_k = \vec{q}_k - \vec{q}_{k-1}$$

Δ_k basically tells in which direction the algorithm is moving. The angle between the last two directions is

$$\theta_k = \operatorname{acos}\left(\frac{\Delta_k \cdot \Delta_{k-1}}{\|\Delta_k\| \, \|\Delta_{k-1}\|}\right)$$

The idea behind extrapolation is, that if the algorithm moved in largely the same direction during the last few iterations, as is the case when both θ_k and θ_{k-1} are small, it may just as well make greater moves.

If the angles are small, extrapolation might be considered. The overall goal is to minimize the value of the objective function, E_k, so a regression over Δ_k vs E_k should be done. One possible outcome is depicted in figure 3.4.

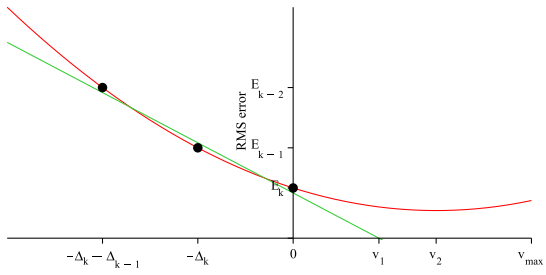


Figure 3.4: Extrapolation based on the relative change in the transformation state vector. In the situation depicted, the zero crossing of the linear interpolant, v_1 would be chosen as the extrapolating factor.

It is seen that the decision on how far to extrapolate should depend on the shape of the progression shown in the figure. Denoting the location of the linear extrapolants' zero crossing v_1 and the location of the parabolic extrapolants' top point v_2, a good strategy is to chose which ever is smaller in magnitude to avoid shooting over the target. Additionally a maximum bound on the extrapolation length, $v_{\max} = 10 \|\Delta_k\|$ provides a sanity check.

Having determined the magnitude of extrapolation, a new transformation state vector, $\vec{q}_k$, is given by:

$$\vec{q'}_k = \vec{q}_k + v\frac{\Delta_k}{\|\Delta_k\|}$$

Finally the quaternion part of $\vec{q'}_k$ is normalized to yield a rigid transformation.

4 Methods

The evaluation of the surface registration algorithms should be made on data that represents the real scanning situation in the HRRT. However the method for tracking the position of the face and the processing of the images into a point cloud is not a simple task. These steps introduces many uncertainties and variables to the data that could complicate the analysis and question the validity of the results.

This chapter presents some of these challenges and issues and describes how we address them with certain equipment, setups, techniques or simplifications.

Experimental considerations The structured light scanner itself is a challenge since it has numerous settings and configurations. The system is described in section 4.1.

A ground truth should be obtained if possible. This project is based on the assumption that all motions can be considered rigid. The experiments had to be made, so that this assumption holds as true as possible. By using a mannequin head that rotates on a motorized axis, the assumption of rigidity holds and a ground truth is known. The mannequin and motor system is described in section 4.2.

Just working with a mannequin head does not reflect the realistic situation. We have to explore the performance of the algorithms on real faces as well. This presents the problem of not being able to know the exact transformations that are performed. A reference system should be used, and this could be the Polaris Vicra system or anatomical landmarking technique, both of which are presented in section 4.3.

Data processing considerations The algorithms for point cloud reconstruction can be a source of noise and errors as well. This is addressed in section 4.4.

The experiments that we performed and base our evaluation on can be grouped into two categories:

Type 1: A real face or mannequin head captured in several different positions with variations in all six degrees of freedom. The exact transformations are not known, and the Polaris Vicra system was used as a reference.

Type 2: A mannequin head, that was attached to a programmable motor with one axis of rotation, was rotated and captured in different positions. Here the exact transformations are known.

All experiments were conducted at the PET and Cyclotron Unit at Rigshospitalet (Copenhagen University Hospital).

4.1 Structured light scanner:

The most important piece of equipment was obviously the system that was used to obtain the point clouds, the structured light scanner (SLS). The system was a work in progress and could be composed with different hardware parts in several different ways. The system configuration that we base our evaluation on is shown in figure 4.1.

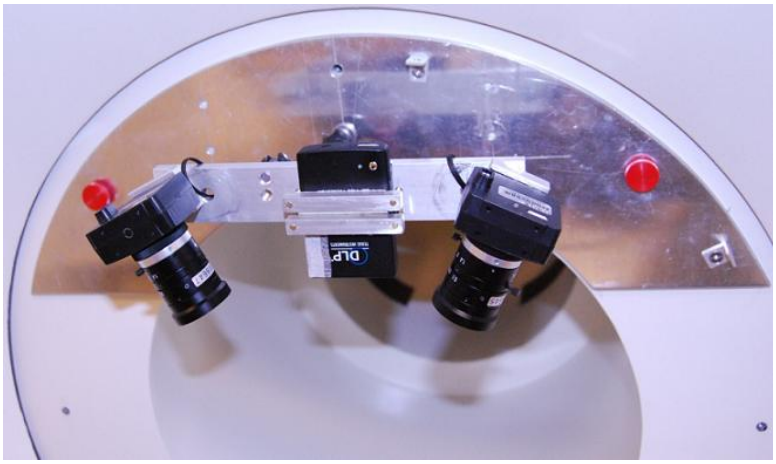


Figure 4.1: The structured light scanner mounted on the HRRT

- The projector in the center is a DLP Pico Projector Development Kit version 2 from *Texas Instruments*.

- The two cameras are Chameleon^{TM}CMLN-13S2C 2.0 USB digital cameras from *Point Grey*. The cameras were mounted with 6 mm lenses with aperture 1.4 from *Pentax*.

For each registered point cloud a total of five light patterns were projected and captured by the cameras. The patterns and the sequence are seen in figure 4.2.

(a) Pattern 1 **(b)** Pattern 2 **(c)** Pattern 3

(d) Pattern 4 **(e)** Pattern 0

Figure 4.2: The sequence of the light patterns. Pattern 1 to 3 are shifted versions of the same grid. Pattern 4 is just bright white light, and pattern 0 is three vertical lines that are used to define the center for point cloud reconstruction.

There were several challenges regarding the use of the SLS, since it was not a finalized system. Instead of being based on visual light, the system could be configured with near infra red (IR) light, or the camera lenses could be changed etc. All these alterations could have an impact on the quality, size and shape of the data and therefore an impact on the registration error. This however would introduce a lot of extra variables to the evaluations. For the sake of simplicity we therefore limit our evaluation to be based only on the one specific SLS configuration. Another challenge is the real capturing situation. The patient tunnel of the HRRT is very narrow and in a real scanning the patient is placed far into the tunnel. The cameras tend to get a really good line of sight to the nostrils, which is highly unfavorable since the forehead is an area of interest as described in section 1.3.

4.2 The mannequin head and the motor

The setup with a mannequin head on motor is shown in figure 4.3.

Figure 4.3: The mannequin head and the motor.

The mannequin head was made of a plastic material. It was completely symmetric, and the surface was much smoother than real skin, which provides point clouds with less noise. The point clouds contains less noise.

The head was attached to a motor from *Thorlabs*. The motor had one axis of rotation, and could be programmed to rotate stepwise or continually from 0 to 360 degrees. The velocity and acceleration of the rotation and dwell time at each step could also be controlled.

The setup is useful because the transformation is rigid, and because it gives accurate and precise control over the motion. It is however limited a single axis. The resulting point clouds are relatively noise-free, and all these good conditions makes it possible to use this setup in more challenging situations, such as recordings of the mannequin head inside the scanner.

4.3 Reference system

A reference system is another system that can be used to evaluate the estimated transformations. This could be the Polaris Vicra system or the anatomical landmarking technique.

Anatomical Landmarking

With the use of a software tool, the same anatomical landmarks are manually placed on every frame. This should be immobile points on the face, such as the nasion. The landmarks serves as points with a known correspondence. When a transformation has been found, the same transform is applied to the landmarks on the data point set. The distance between the model and data landmarks serves as a measure of the error of the transformation.

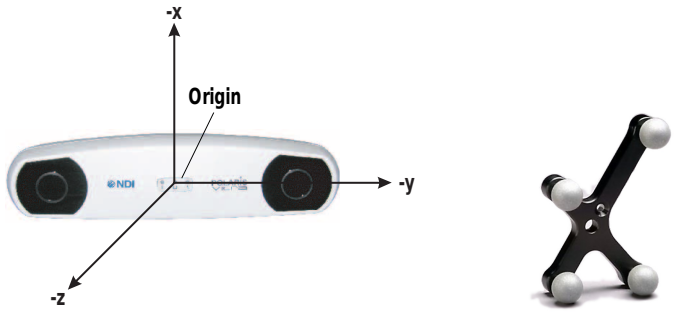
The advantages with this technique is that it does not require any further experimental preparation. The disadvantages is that the landmarks should be placed at immobile locations. There are not many of those on the human face and there is a risk that these points cannot even be seen on the frame. This technique also requires training to able to achieve high precision and accuracy.

Polaris Vicra

The Polaris Vicra system from *NDI* is an infrared based system to track and measure 3-D positions of retroreflective markers. The system consists of a *position sensor* and a *tracking tool*, both of which are shown in figure 4.4. The tool is an object with four spherical markers attached in a unique 3-D relation.

The position sensor can keep track of the tool within a pyramidal *measurement volume*, by emitting IR-light and receiving the reflected light from the markers. The approximate extend of the measurement volume is shown in figure 4.5. Based on this information the system calculates the position and rotational state of the tool and saves it in a quaternion representation.

The system has a specified accuracy of 0.35 mmRMS [39]. However the HRRT is a very narrow scanner. From the experiments we know that the measurement volume is just large enough to engulf the patients tunnel of the HRRT when the sensor is placed right behind the scanner. Establishing line of sight from all markers on the tool to the position sensor can be cumbersome. It is also



(a) (b)

Figure 4.4: (a): The Polaris Vicra position sensor. The negative basis vectors of the sensors coordinate system are shown. Source: Polaris Vicra User Guide [19]. (b): A rigid body tool with four retroflective markers.

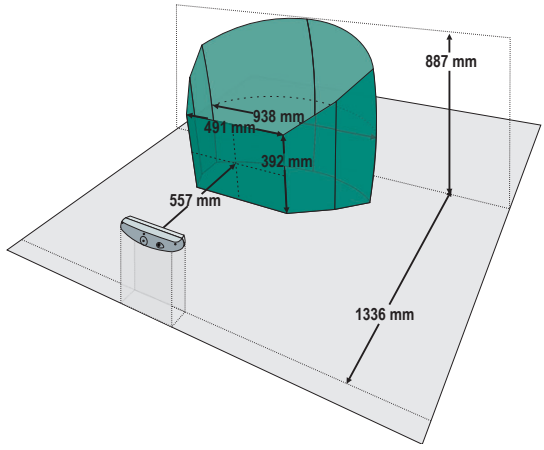


Figure 4.5: The measurement volume of the Polaris Vicra system is shown in green. Within this volume the position and rotational state of the tool can be determined. Source: Polaris Vicra User Guide [19].

important that the tool is fixed well to the patient. The in order to compare results of the Polaris Vicra system with registration results, we employed the method described in appendix B.

4.4 Data preprocessing

All images were taken through several processing steps. These steps are shown in figure 4.6. The computational time and complexity of the preprocessing is ignored when evaluating the ICP algoritms.

Point cloud reconstruction:

Each structured light frame was converted into point clouds based on the fast two-dimensional phase-unwrapping algorithm developed by [16]. The reconstruction can however encounter severe problems, where areas of the face are placed in an obviously wrong position. These faults can be solved manually by forcing the area to the correct position or by excluding the area.

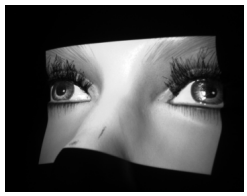
Our evaluation is based on the assumption that no point clouds contain any of these major errors.

Combining point clouds:

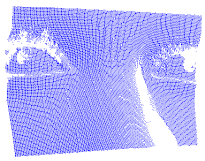
The structured light system has two cameras. Combining the point clouds from both cameras gives a few advantages. First of all the amount of available frames is greatly reduced, and then the evaluation does not need to distinguish between cameras. Secondly, the alignments are more likely to succeed, since there will be more overlapping area.

The combining is not just adding all the points together. That would not work – since overlapping areas of the point clouds would have much higher point density than non-overlapping areas. As a consequence the surface reconstruction would misjudge the correct point density and give erroneous results. Instead the combining is done by using an ICP with a modification. First an alignment with edge rejection is performed. All points from the data set that have neighbors in the model set closer than a given threshold are removed. The transformation of the point cloud is not recorded. This will obviously result in an uncertainty in

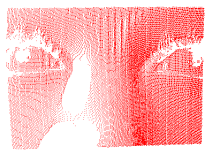
(a) Camera 1



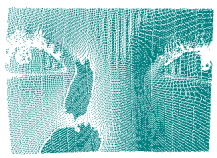
(b) Camera 2



(c) Point cloud reconstruction of camera 1



(d) Point cloud reconstruction of camera 2



(e) Combined point clouds

(f) Surface reconstruction

Figure 4.6: The steps in preprocessing of data. a) and b): The illuminated image from each camera. c) and d): Each set of images are reconstructed into a point cloud. e): The two combined point clouds. f): The surface reconstruction.

the evaluation results. But if the point cloud reconstruction is done properly, which we assume it is, then the two clouds will have a good overlap to begin with. Than it is safe to assume that this initial registration does not move the data point cloud much, and the model point cloud remains stationary. The advantage is a reduction of noise and better surface reconstructions.

Surface reconstruction:

Surfaces were reconstructed for all point clouds using the MRF surface reconstruction software package version 1.0 [31]. The software provides a triangulation, normals and gives a point cloud with more uniform point density.

After this step then we have decent, relatively noise-free surface reconstructed point clouds to work with.

5 Experiment I - Convergence

One point cloud was used to compare convergence behavior of the ICP variants isolated from other factors. The points were transformed using a known transformation, and Gaussian noise was added independently to both point clouds. This is the same approach as used by Rusinkiewicz and Levoy [34] adopted specifically to a point cloud of the face. Comparisons are based on the root mean square of distances between the *actual* point pairs. The test scene is seen in figure 5.1.

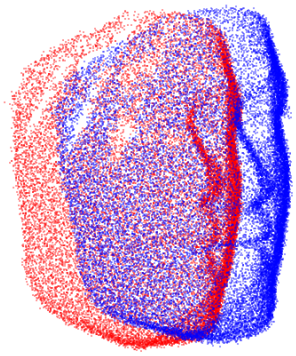


Figure 5.1: Point cloud of the face with added Gaussian noise (red). The transformed point cloud with independent Gaussian noise (blue).

Table 5.1 shows the ICP features that were assessed in every taxonomy class. The classes were investigated independently while all other features were set to their defaults.

5.1 Selection

The results of different selection methods are shown in figure 5.2. In all cases, a fraction of 1/50 of all points were used.

In normal-space sampling, the points were put into a total of theoretically 27 buckets.

Taxonomy class	Feature
Selection	
	{ All points }
	Uniform sampling (1/50th of all points)
	Random sampling (1/50th of all points)
	Normal-space sampling (1/50th of all points)
	Curvature-space sampling (1/50th of all points)
Matching	
	{ Nearest neighbor }
	Normal shooting
Weighting	
	{ None }
	Curvature compatibility
	Normal compatibility
	Linearly with distance
Rejection	
	{ None }
	Worst pairs percentage
	Edge vertices
Error metrics	
	{ Point to point }
	Point to plane
	Point to point with extrapolation
	Levenberg-Marquardt
Minimization	
	{ SVD }

Table 5.1: ICP variants under investigation. Default values are enclosed in curly brackets.

Curvature-space sampling was done using Pauly's curvature estimate based on 100 nearest neighbors. Points were then sampled from 10 buckets of equal range.

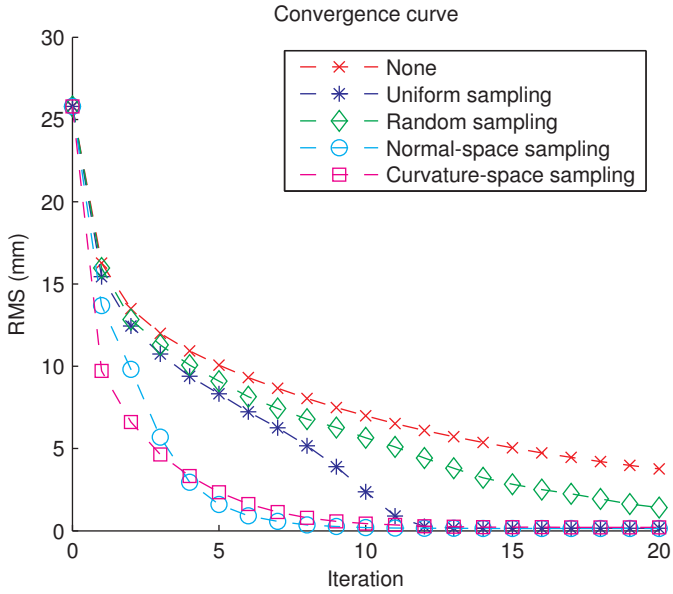


Figure 5.2: Convergence behavior using different sampling methods. In all cases, 1/50 of the available points were used.

5.2 Matching

Two point matching strategies were investigated, nearest neighbors and the ray shooting as described in section 3.2.2. The results are shown in figure 5.3.

5.3 Weighting

Point pairs were weighted using the normal compatibility, curvature compatibility and linear distance weight functions described in section 3.2.3. Curvatures were estimated using Paulys method with 100 nearest neighbors. The results are shown in figure 5.4.

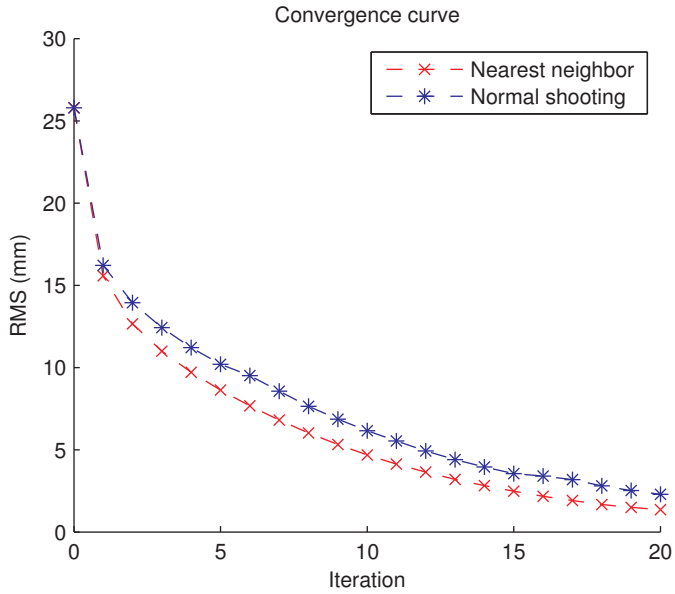


Figure 5.3: Convergence for the matching strategies nearest neighbors and *normal shooting*.

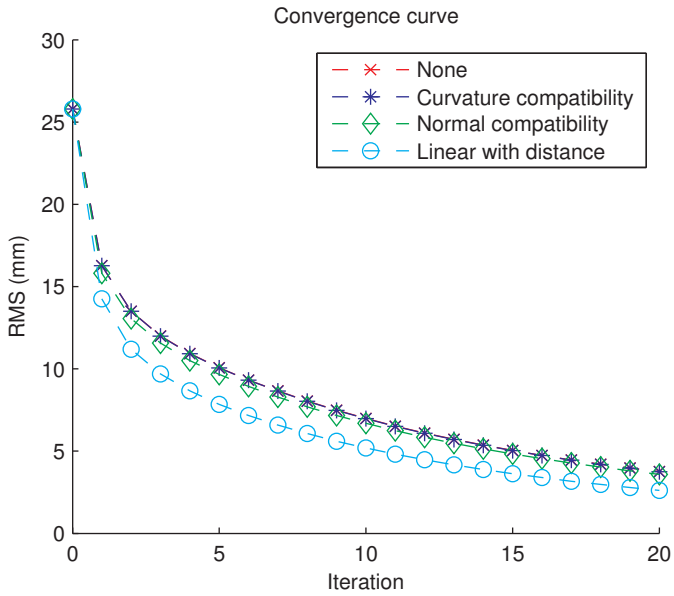


Figure 5.4: Convergence for different weighting schemes.

5.4 Rejection

The edge rejection and 10% worst pair rejection schemes were compared to the baseline in figure 5.5.

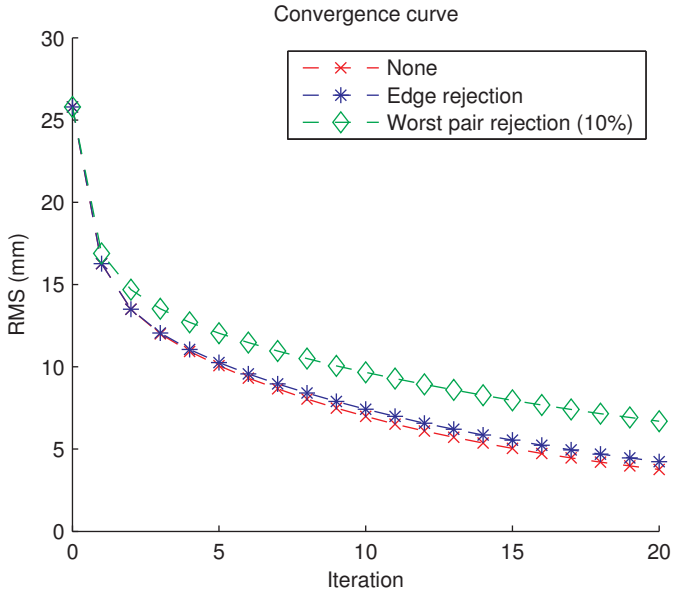


Figure 5.5: Convergence behaviour using different rejection methods.

5.5 Error metric

The error metrics in consideration were point to point and point to plane. Additionally point to point minimization with extrapolation was performed. These were compared to the Levenberg-Marquardt algorithm for a point to point error metric. The results are shown in figure 5.6.

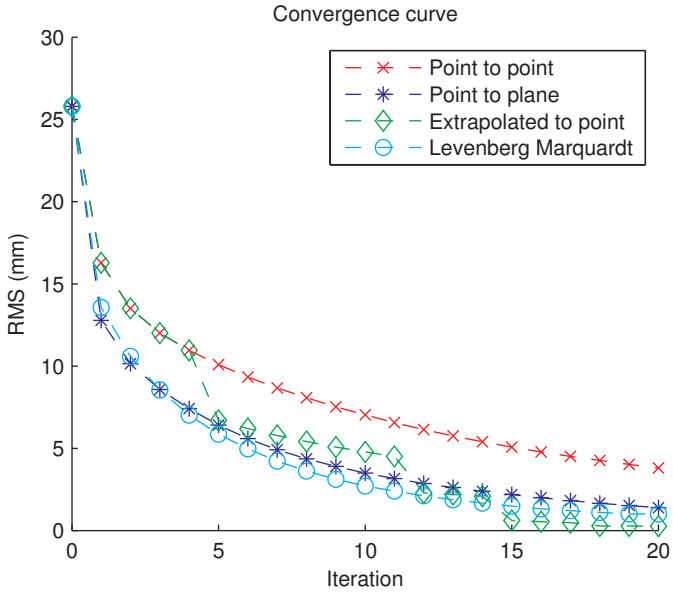


Figure 5.6: Convergence behaviour using different error metrics and compared to Levenberg-Marquardt minimization. The extrapolated iteration steps are seen as jumps in the green convergence curve.

5.6 Discussion

Clearly, the choice of selection strategy has a massive impact on convergence. Specifically normal-space and curvature-space sampling give pleasing results with relatively simple implementation. They represent simple methods of feature extraction – the parts of the point clouds that represent anchoring points, like the canthus, will be represented well. The downside of both forms of sampling is the added preprocessing time for estimation of surface normals or curvature. Uniform selection performs well, and definitely better than random selection. This might be explained by the regularity in the surface – uniform

selection keeps a regular point distance, while this is not guaranteed with random sampling. We notice that using all points yields the worst results but is expected to be the most robust choice.

It is seen that nearest neighbors matching generally performs better than normal shooting. One should keep in mind though that normal shooting can give better results in the presence of outliers, however it requires the initial alignment to be good, and also it requires good surface normals. The initial alignment in the test scene is worse than can be expected in a real time ICP application, and this might explain these results.

The proposed weighting strategies have little effect on the convergence. This is in accordance with the observations of Rusinkiewicz and Levoy [34]. Weighting can therefore not be recommended for our purposes, it only adds to the complexity of the algorithm.

In rejection strategies, it should be noted that these methods are expected to have their greatest impact on point clouds with partial overlap. The added robustness comes at the price of decreased convergence. We will not dismiss rejection strategies and consider them in the next experiments.

Matching strategies are seen to have a big influence on convergence. The Chen-Medioni approach of using point to plane minimization gives considerably better results than the Besl-McKay framework with point to point minimization. The Levenberg-Marquardt algorithm has monotonic and smooth convergence that is a little better than point to plane. A most interesting find is, that point to point minimization with extrapolation in the end is better than all other minimization strategies. Since the implementation effort is low, it is highly recommended.

6 Experiment II - Accuracy

It is difficult to assess the accuracy of point cloud registration without referring to a ground truth. Comparing to the Polaris Vicra system is possible, but it introduces its own measurement uncertainty. We settle for the next best by comparing the results obtained by the algorithms with a manual alignment that gives the best visually perceived results. Our objective is to find an alignment that is invariant towards deformations of the face between scans and robust against noise.

We will give results obtained from the Polaris Vicra system only as a reference.

6.1 Procedure

The subject laid on the examination table of the HRRT with his head positioned on a neck rest outside the scanner gantry. The Polaris sensor was placed behind the scanner, so that line of sight was established at all times through the scanner gantry. The tracking tool was attached near the hairline of the subject using a Velcro band-aid [28] – see figure 6.2. This setup allowed for a good SLS camera angle while keeping a near realistic recording scenario.

Initial position: The subject was facing upwards. This forms the initial frame to which all subsequent frames were registered.

Patient movements: The subject then repositioned his head into six different positions.

At each position a frame was captured with the SLS, giving a total of seven frames. The Polaris system was triggered manually at every step, and the positions corresponding to each frame were later extracted from the recordings. The angle of rotation between the initial position and the subsequent six positions shall be denoted $\theta_1, \theta_2, ...\theta_6$. The SLS field of view covered only a part of the subjects face. The regions of overlap between frames were around 80% of the field of view.

Surface reconstructions were made from the point clouds, smoothening them and removing outliers.

6.2 Results

The visually guided alignments are seen in figure 6.1. Table 6.1 shows the angles of rotation as found through the manual alignment and using the SLS and point cloud registration. The settings used are given in the row titles of the table.

6.3 Discussion

It is clearly seen, that the standard ICP variants, point to point and point to plane, yield results that far from the manual alignment. This does not surprise, since the point clouds in question have only partial overlap. The algorithm will simply make them overlap as much as possible resulting in a bad alignment.

Applying the normal shooting matching method as the only difference gives equally bad results. In two cases, θ_2 and θ the algorithm fails completely. In these cases, the initial alignment was six to nine degrees off, which apparently is more than needed by the normal shooting method.

Rejecting the ten percent worst point pairs has a positive effect on alignment results. Generally this allows the algorithm to align to 90 % partial overlap without penalty.

In the edge rejection cases, results are close to the manual alignment. While point to plane ICP and the Levenberg-Marquardt converge to the same minimum in most cases, the point to point ICP does not reach quite that point. Since this variant generally converges slower than the other, twenty iterations might simply not be enough for it.

The Polaris Tool Tracker measured angles that are far from the manual aligned, and they are not considered valid. The reason for their large offset might be the lack of a stable bond to the head during recordings. The aforementioned fixation to the forehead of the test subject (figure 6.2) proves unreliable.

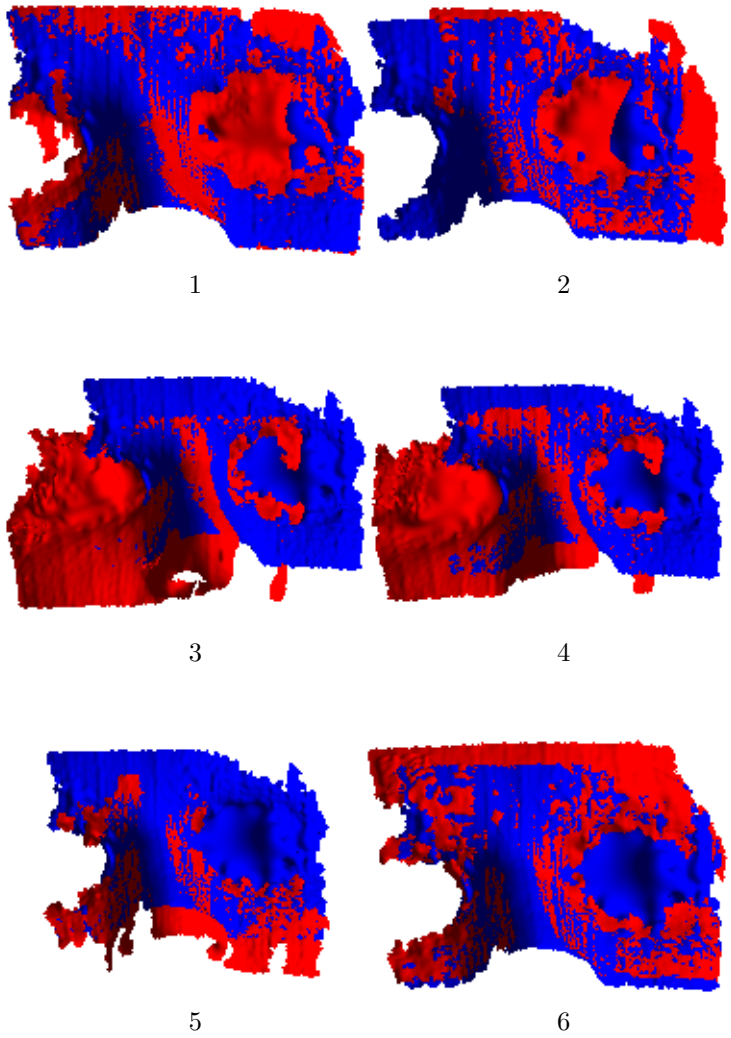


Figure 6.1: The visually *best* alignments of six point clouds (red) to the reference frame (blue). Regions of blended colors indicate good alignment. The point clouds were preprocessed by MRF surface reconstruction, which allows for a shaded surface rendering.

θ_1	θ_2	θ_3	θ_4	θ_5	θ_6
Manual alignment					
0.90	7.07	8.52	7.13	5.37	2.20
Point to point ICP					
0.90	5.63	18.53	10.45	6.20	1.30
Point to plane ICP					
1.36	6.55	9.51	8.33	5.66	1.57
Point to plane ICP with normal shooting					
1.30	180.00	140.02	9.00	4.77	2.06
Point to plane ICP with 10 % rejection					
0.49	6.65	11.41	9.01	5.04	1.88
Point to point ICP with Edge Rejection					
0.98	6.25	13.46	6.79	3.74	1.79
Point to plane ICP with Edge Rejection					
1.02	6.48	9.41	7.60	5.12	1.85
Levenberg Marquardt with Edge Rejection					
0.52	6.45	9.40	7.54	5.02	1.87
Polaris Tool Tracker					
6.96	11.43	9.67	4.53	2.53	2.88

Table 6.1: The resulting angles from six ICP alignments compared to the manual alignment and the angles measured using Polaris Vicra.

Figure 6.2: The head of the human subject for experiment II with the Polaris rigid body tool attached using a Velcro band-aid – see Olesen 2009 [28].

7 Experiment III - Robustness

Robustness is a measure of how well the algorithms can withstand noisy data. In also assesses how little true overlapping area the algorithms need in order to find a correct alignment of the point clouds.

One way to depict robustness is by finding the basin of convergence (BOC). In this context the BOC is the alignment error as a function of how much the clouds have been moved away from each other. However we simplify it to be the alignment error as function of a rotation angle – since this is an experimental possibility. In order to make a BOC it is required to know a ground truth and to have the ability to accurately control transformations. This is very complicated to achieve experimentally in all six degrees of freedom. The simplified version is however possible with our experiment with the mannequin attached to a motor.

7.1 Procedure

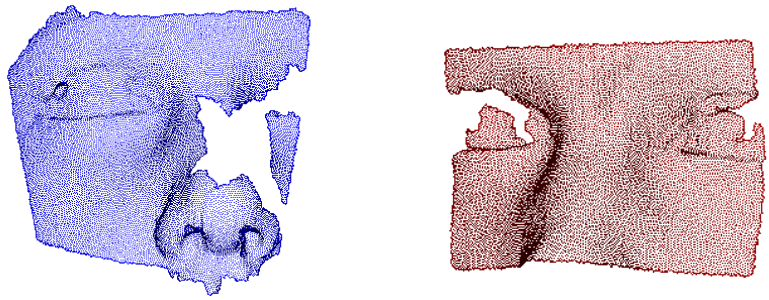
The mannequin head with the motor was placed on the examination table, with the head inside the patients tunnel. The position where the mannequin was facing straight upwards was defined as a zero degree rotation. The motor was programmed to rotate to -20 degrees relative to zero. From here it was rotated stepwise to $+20$ degrees, with a step size of 2 degrees. At each step a registration was made – which sums up to a total of 21 recordings.

This procedure was repeated a couple of times (runs). One run was made where the mannequin was lying with the head outside the scanner for a more optimal recording condition.

The ICP alignments are made with a single reference frame to which all frames of the run are aligned to. A center frame (zero degree rotation) from another run is designated as reference, so that the center frame is not aligned to itself. Except for the run where the mannequin was outside the scanner, here all the frames are aligned to the zero degree frame of that run. The RMS error for all variants without edge rejection are modified so that the RMS error is actually only calculated for overlapping areas. This is to avoid a bias towards methods using edge rejection.

7.2 Results

The reference frame is shown in figure 7.1. It is seen that image is not symmetric around the nose.



(a) (b)

Figure 7.1: The two reference frame used. Left: The reference frame used in figure 7.2. Right: The reference frame for the results in figure 7.3

BOC shown in figure 7.2:
Point to plane: Both methods are equally good when angles are no more than ± 10 degrees, but edge rejection gives a slight performance boost beyond that. The upper basin boundary is $+16$ degrees for both. The lower boundary is not sharply defined as the upper bound. Without edge rejection the alignment error steadily worsens. With edge rejection the RMS varies. Inspecting the deviations from ground truth angles is seen in table 7.1.

Ground thruth (degrees)	-20	-18	-16	-14
With ER (degrees)	1.3	0.9	1.4	1.1
Without ER (degrees)	8.7	5.9	2.2	1.2

Table 7.1: Deviations from ground thruth

Point to point: The variant with edge rejection manages to match both point to plane methods until -12 and $+14$ degrees.
The variant without edge rejection is not even shaped as a basin. The RMS error steadily worsens for increasing angles.
Levenberg Marquardt: Has a clearly defined basin of convergence from -14

to +16 degrees. Within this range is performs equally to point to plane with edge rejection.

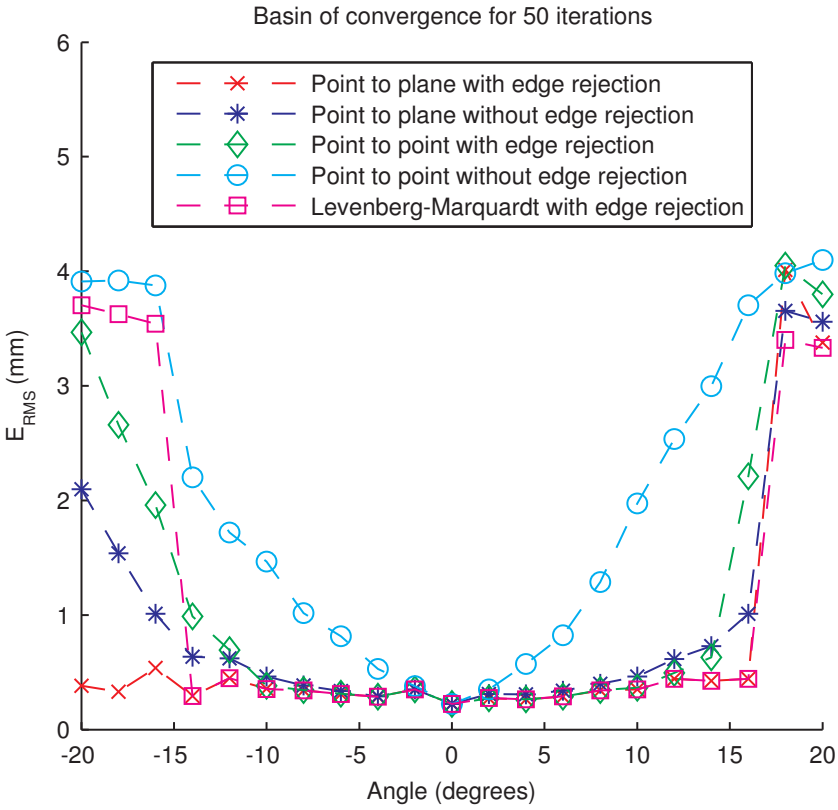


Figure 7.2: Basin of convergence for different algorithm variants. The mannequin head was located inside the HRRT gantry.

BOC shown in figure 7.3:
Point to plane: The variant without edge rejection steadily worsens until it reaches the boundaries at ±14 degrees.
The basin boundaries are almost the same for variant with edge rejection. This variant however has a low and constant RMS error until it reaches the boundaries.
Point to point: The variant with edge rejection performs almost like the point to plane with edge rejection.
The other point to point variant now uses worst rejection. It perform much better than it did in figure 7.2. The edge rejection is however even more robust.
Levenberg-Marquardt: Is almost identical to the variant point to plane with

edge rejection.

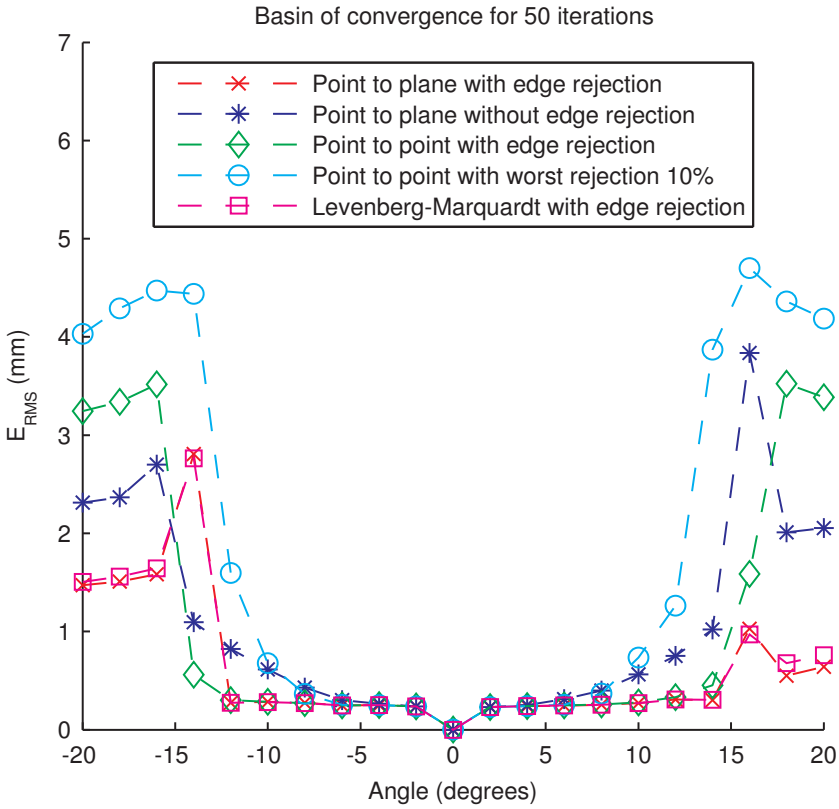


Figure 7.3: Basin of convergence for different algorithm variants. The mannequin head was located outside the HRRT gantry. The 0 degree alignment was done with the reference itself, thus giving 0 mmRMS in all cases.

Discussion

Ideally a basin of convergence should be symmetric. If this is not the case then there is some kind of bias. The second BOC (figure 7.3) is as symmetric as we could expect, but the first (figure 7.2) clearly favors negative angles. The reason is the reference frame. The mannequin must have been facing a bit to one side in the zero degree position. As a consequence of this bias it is possible for the point to plane variant with edge rejection to find pretty accurate results even at the biggest angles, as shown in table 7.1. However it achieves lower RMS error

and higher angle accuracy at -20 and -18 degrees than at -16 degrees. This could indicate that it is more about luck than actual algorithm robustness.

We expected that rejection techniques would have a major influence, since we work with point clouds with partial overlaps. Worst rejection improves the robustness as we expected, but the edge rejection is just better. It can be seen that the point to point variant is close to useless without a rejection technique. However the point to plane variant did well without at smaller angles. But it is also clear that point to plane benefits from having edge rejection.

The non-linear Levenberg-Marquardt algoritm is seen to be as robust as the ICP.

8 Experiment IV - Speed

As in most algorithmic problems, computation speed is a point to consider. Performing point cloud registration for motion correction in a clinical work flow should be fast – preferably real time. It was pointed out, that the computational expensive part of the ICP algorithm is the matching step. This in turn is dependent on

1. the number of points after sampling
2. the choice of data structure and algorithm
3. the implementation

This experiment compares differences in point number 2.

8.1 Procedure

We use a single test scene of the mannequin head which is shown in figure 8.1.

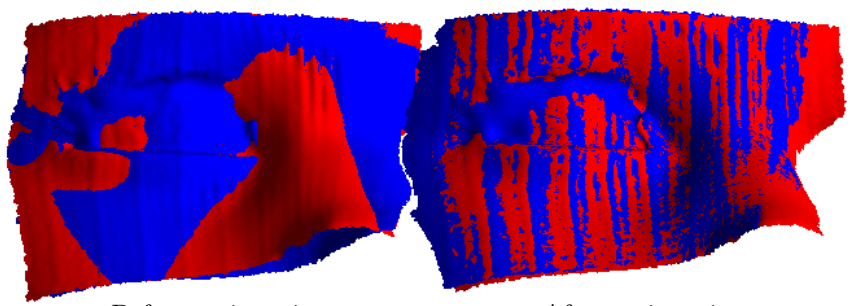


Figure 8.1: Test scene used for experiment IV before and after manual transformation. The blending of colors indicate a good match. The streaky pattern is due to errors in the SLS.

In our approach to quantify the performance of algorithms, we compare among the matching strategies. The point clouds contain approximately 15 k points,

and no sampling is done. Our implementations use the precompiled Matlab functions for kD trees, Delaunay Triangulations and brute force search.

All variants include edge rejection.

Registrations are done with 50 iterations, and computational times are measured using Matlabs stopwatch functions. All measurements were done on a modern Linux computer using Matlab 2010a.

The RMS error is based on the distances from data points in a given iteration to the same points in the manual alignment in figure 8.1. Loosely speaking, this RMS error measure tells how far we are from where we want to be.

8.2 Results

Figure 8.2 shows the iteration count as a function of time since the algorithm was started.

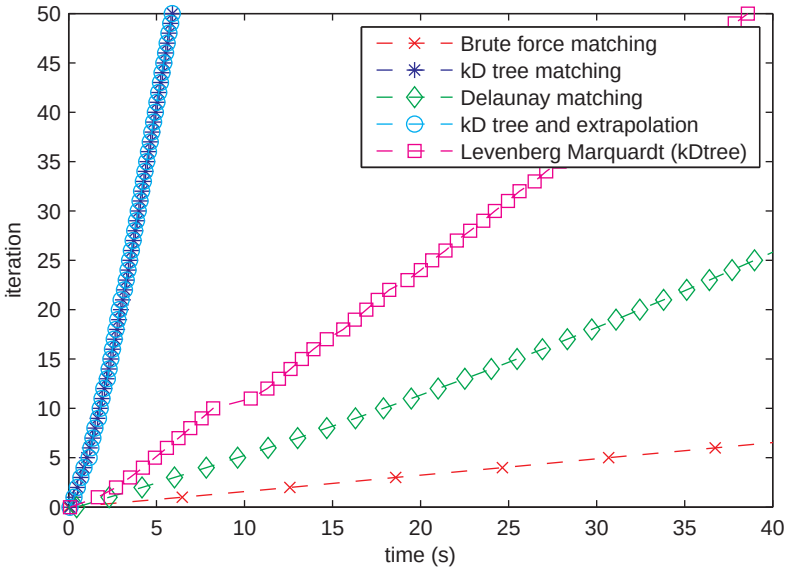


Figure 8.2: The iteration count as a function of time. Slight offsets indicate precomputation times for data structure creation.

It is seen that all variant perform near linear. kD tree variants are the fastest.

There is no significant difference with extrapolation. Iterations take on average 0.15 seconds.

The Levenberg-Marquardt algorithm is used with kD tree accelerated neighbor searching, and it is much slower in this scenario with approximately 1.1 second per iteration. The computation time per iteration is seen to vary for this algorithm.

Delaunay triangulation and brute force matching strategies are slowest with 2 seconds/iteration and 6 seconds/iteration respectively.

In figure 8.3 the RMS error of the known point correspondence is shown against computation times. The curves integrate the iteration time measures with convergence curves.

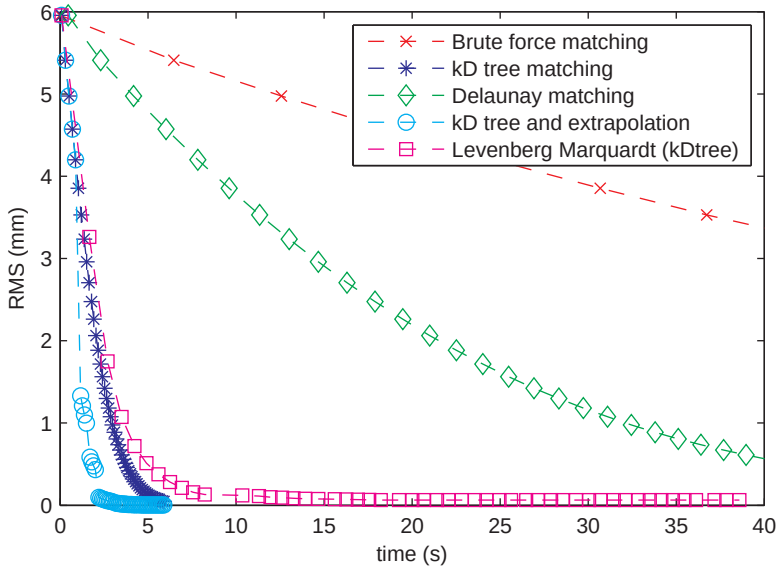


Figure 8.3: The RMS error as a function of time. The symbols indicate iteration counts.

It is seen that all variants converge monotonically with respect to our RMS error definition. The kD tree matching variants with and without extrapolation follow along until iteration four, where an extrapolation occurs. The extrapolating algorithm converges completely in about three seconds.

The Levenberg-Marquardt optimizer converges very quickly in the first iterations, but slows down significantly at RMS under 1 mm.

Delaunay and brute force matching strategies find exactly the same neighbors as the kD tree matching method – only slower. Their time convergence curves are thus what is to be expected from the time iteration curves in figure 8.2.

8.3 Discussion

Not surprisingly, the most simple ICP variants using kD trees perform best in terms of time per iteration. The computational overhead caused by extrapolation is neglectable. It does on the other hand give significantly quicker convergence overall.

Interestingly, the Levenberg-Marquardt algorithm finishes as a close candidate. It does however suffer speed wise in later iterations.

Delaunay and brute force matching are not competitive and can be regarded unsuitable for this kind of registration problem.

Implementation details taken aside, it can be concluded that for point cloud registrations of the face with quite good initial alignment, heuristic approaches to the correspondence problem are inevitable. Our theoretical discussions in section 2.3 indicate, that this conclusion holds also when subsampling is done. We stress the benefit of Besl-McKay extrapolation, and recommend it for eventual commercial implementations.

With our implementation, we have shown that computational time of the ICP can be as low as three seconds for a unsampled point cloud.

If real time performance is the objective, an even more efficient algorithm would be necessary. We suggest the following for further investigations:

- Subsampling. We showed that convergence is better with the right sampling strategy. The added benefit is reduced computation time.
- Optimization of the matching step. A simple modification would be in early iterations to do *approximate nearest neighbor* searches, as described in section 2.3.
- Reverse calibration as described in section 3.2.2.
- Multi resolution ICP – going from coarse to fine registrations as described by Jost and Hügli [21].

- A compiled implementation in a lower level programming language. Preferably with parallel processing. The ICP algorithm is also suitable for a GPU implementation [22].

9 Conclusion

The goal of this thesis was to evaluate the applicability and efficiency of surface registration algorithms for use in a structured light motion tracking system. The input data were partial point clouds of the face.

Our evaluation was based on the assumption that all motions were rigid and that the structured light system provided consistent point clouds without major defects. We focused on the performance regarding convergence, accuracy, robustness and speed, which were analyzed separately.

One of the biggest concerns with data of this kind is partial overlap. It was seen quite clearly, that ignoring this leads to inaccurate results even at small changes in posture. The most sophisticated solution to this is edge rejection, and it effectively solves the problem. Rejecting the worst percentage of point pairs can effectively do the same and is possible even when a surface reconstruction is not available.

The major hurdle regarding the speed of the algorithms was neighbor searching in the point clouds. We have shown that space-partitioning is highly favorable over other methods. Methods that can improve the rate of convergence will also reduce computational times as a consequence. We have shown that proper sampling in normal- or curvature-space and the use of extrapolation do just this very effectively. Our discussion of the speed results lead to the conclusion that real time performance is a realistic perspective.

The Levenberg-Marquardt optimizer is suggested as an alternative if one needs full control over the objective function. It also has the advantage of being efficiently implemented in many software packages. Since these are not the real issues for a real time implementation, we do favor the ICP algorithm for its higher speed.

A Singular Value Decomposition

The derivations of the point to point minimization given in appendix C requires knowledge of a popular matrix matrix factorization called singular value decomposition (SVD). The SVD is commonly met in linear minimization problems.

Whilst related to the *eigendecomposition*, SVD exist for every two-dimensional matrix, including complex ones. Because all matrices encountered in this report are real it is assumed that in the following, $\mathbf{M}$ denotes a real, $m \times n$ matrix. For such a matrix the SVD is given by

$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices of size $m \times m$ and $n \times n$ respectively. Recall that orthogonality means $\mathbf{U}^{\mathrm{T}} = \mathbf{U}^{-1}$ and $\mathbf{V}^{\mathrm{T}} = \mathbf{V}^{-1}$. $\mathbf{\Sigma}$ is a diagonal $m \times n$ matrix containing the *singular values* of $\mathbf{M}$.

Consider the matrix product $\mathbf{M}^{\mathrm{T}}\mathbf{M}$. Using the above, this may be written

$$\mathbf{M}^{\mathrm{T}}\mathbf{M} = \mathbf{V}\mathbf{\Sigma}^{\mathrm{T}}\mathbf{U}^{\mathrm{T}}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}} = \mathbf{V}\mathbf{\Sigma}^{\mathrm{T}}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}} \tag{A.1}$$

Clearly, this is an eigendecomposition in which the columns of $\mathbf{V}$ hold the eigenvectors of $\mathbf{M}^{\mathrm{T}}\mathbf{M}$. The product $\mathbf{\Sigma}^{\mathrm{T}}\mathbf{\Sigma}$ contains the eigenvalues, which are the squared singular values of $\mathbf{M}$. Equivalentely the columns of $\mathbf{U}$ are eigenvectors for the product $\mathbf{M}\mathbf{M}^{\mathrm{T}}$. The existence of an SVD for every rectangular matrix is implied by equation A.1. On the other hand, a given SVD is not unique because the order of singular values in the diagonal of $\mathbf{\Sigma}$ may change. A common convention is to let the absolute value of singular values descend over the diagonal.

The columns of $\mathbf{V}$ and $\mathbf{U}$ are called the left and right *singular vectors* of $\mathbf{M}$.

B Coordinate alignment

A registration in Polaris consist of a unit quaternion and a translation vector that describe the rigid transformation of the measurement tool from some fixed position in space. In order to find the relative rotation q_r between two quaternions, q_1 and $q_2 = q_r q_1$, one takes

$$q_r = q_2 q_1^{-1} = q_2 q_1^*$$

utilizing the fact that the inverse of a unit quaternion is just its conjugate. The quaternion representation can then be converted to the axis angle representation as described in appendix E.

The SLS operates on its own coordinates, and the result of a point cloud registration is the relative transformation between two frames expressed in a rotation matrix that can be converted to the axis angle representation using techniques described in appendix E.

Now rotation angles can be compared directly, since their magnitude is independent of the coordinate system in question.

C Point to point minimization

The following describes least squares minimization of the alignment error using point to point minimization.

Let p_i and q_i denote the N matched point pairs. The normals corresponding to the model points are denoted $\vec{n}_i$. The centroids are defined as

$$\bar{p} = \frac{1}{m}\sum p \qquad \bar{q} = \frac{1}{n}\sum q$$

where m and n are the number of model points and data points respectively.

The points deviations from the centroid are given by

$$p'_i = p_i - \bar{p} \qquad q'_i = q_i - \bar{q}$$

The task is to find the rotation matrix $\mathbf{R}$ and translation vector $\vec{T}$ so to minimize the error

$$E = \sum_{i=1}^{N} \left\| \mathbf{R}p_i + \vec{T} - q_i \right\|^2$$

Using the above definitions, this can be rewritten

$$= \sum_{i=1}^{N} \left\| \mathbf{R}(p'_i + \bar{p}) + \vec{T} - (q'_i + \bar{q}) \right\|^2 = \sum_{i=1}^{N} \left\| \mathbf{R}p'_i - q'_i + (\mathbf{R}\bar{p} - \bar{q} + \vec{T}) \right\|^2$$

In order to minimize the error metric, the translation vector $\vec{T}$ should be chosen to move the rotated data centroid to the model centroid

$$\vec{T} = \bar{q} - \mathbf{R}\bar{p}$$

which simplifies the error expression

$$E = \sum_{i=1}^{N} \|\mathbf{R}p'_i - q'_i\|^2 = \mathbf{R}\mathbf{R}^{\mathrm{T}} \sum_{i=1}^{N} \|p'_i\|^2 - 2\,\mathrm{tr}\left(\mathbf{R}\sum_{i=1}^{N} p'_i q_i'^{\mathrm{T}}\right) + \sum_{i=1}^{N} \|q'_i\|^2$$
$$= \sum_{i=1}^{N} \|p'_i\|^2 - 2\,\mathrm{tr}\left(\mathbf{R}\sum_{i=1}^{N} p'_i q_i'^{\mathrm{T}}\right) + \sum_{i=1}^{N} \|q'_i\|^2$$

Now let $\mathbf{N} = \sum_{i=1}^{N} p'_i q_i'^{\mathrm{T}}$. To minimize the error E the trace $\mathrm{tr}\,(\mathbf{RN})$ has to be maximized.

Let the columns of $\mathbf{N}$ and the rows of $\mathbf{R}$ be denoted c_i and r_i respectively where $i \in \{1, 2, 3\}$. The trace of $\mathbf{RN}$ can be expanded

$$\mathrm{tr}\,(\mathbf{RN}) = \sum_{i=1}^{3} r_i \cdot c_i \leq \sum_{i=1}^{3} \|r_i\|\,\|c_i\|$$

where the inequality is just a reformulation of the *Cauchy-Schwarz inequality*. Since the rotation matrix $\mathbf{R}$ is orthogonal by definition, its row vectors all have unit length. This implies

$$\mathrm{tr}\,(\mathbf{RN}) \leq \sum_{i=1}^{3} \sqrt{c_i^{\mathrm{T}} c_i} = \mathrm{tr}\left(\sqrt{\mathbf{N}^{\mathrm{T}}\mathbf{N}}\right) \tag{C.1}$$

where the square root in taken in the operator sense.

Consider the singular value decomposition of N

$$\mathbf{N} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}$$

by choosing the the rotation vector as

$$\mathbf{R} = \mathbf{V}\mathbf{U}^{\mathrm{T}}$$

the trace of $\mathbf{RN}$ becomes

$$\mathrm{tr}\left(\mathbf{V}\mathbf{U}^{\mathrm{T}}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}\right) = \mathrm{tr}\left(\mathbf{V}\mathbf{\Sigma}\mathbf{V}^{\text{-}1}\right) = \mathrm{tr}\left(\sqrt{\mathbf{V}\mathbf{\Sigma}^{\mathrm{T}}\mathbf{\Sigma}\mathbf{V}^{\text{-}1}}\right) = \mathrm{tr}\left(\sqrt{\mathbf{N}^{\mathrm{T}}\mathbf{N}}\right)$$

which according to (C.1) is as large as possible.

D Point to plane minimization

In point to plane minimization, the overall goal is to bring the data points close to the planes in which the corresponding model points reside. Mathematically this can be done by minimizing the dot products of the vectors $\overrightarrow{p_i q_i}$ and normals $\vec{n}_i$. The error metric may be written

$$E = \sum_{i=1}^{N} \left[\left(\mathbf{R} p_i + \vec{T} - q_i \right) \cdot \vec{n}_i \right]^2$$

In order to solve the equation analytically, the rotation matrix must be linearized as given by equation 2.1. This approximation only makes sense for small angles α, as are expected in late iterations of the ICP algorithm.

Using this approximation, the error metric becomes

$$\begin{aligned} E = \sum_{i=1}^{N} \big[& \left(p_{i,x} - \gamma p_{i,y} + \beta p_{i,z} + T_x - q_{i,x} \right) n_{i,x} + \\ & \left(\gamma p_{i,x} + p_{i,y} - \alpha p_{i,z} + T_y - q_{i,y} \right) n_{i,y} + \\ & \left(-\beta p_{i,x} + \alpha p_{i,y} + p_{i,z} + T_z - q_{i,z} \right) n_{i,z} \big]^2 \end{aligned}$$

$$\begin{aligned} = \sum_{i=1}^{N} \big[& \left(p_i - q_i \right) \cdot \vec{n}_i + T \cdot \vec{n}_i + \\ & \alpha \left(p_{i,y} n_{i,z} - p_{i,z} n_{i,y} \right) + \\ & \beta \left(p_{i,z} n_{i,x} - p_{i,x} n_{i,z} \right) + \\ & \gamma \left(p_{i,x} n_{i,y} - p_{i,y} n_{i,x} \right) \big]^2 \end{aligned}$$

Defining

$$c_i = p_i \times \vec{n}_i$$

and

$$\vec{r} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix}$$

the error becomes

$$E = \sum_{i=1}^{N} \left[(p_i - q_i) \cdot \vec{n}_i + \vec{T} \cdot \vec{n}_i + \vec{r} \cdot c_i \right]^2$$

The partial derivatives with respect to the six degrees of freedom are

$$\begin{aligned}
\frac{\partial E}{\partial \alpha} &= \sum_{i=1}^{N} 2c_{i,x} \left[(p_i - q_i) \cdot \vec{n}_i + T \cdot \vec{n}_i + \vec{r} \cdot c_i \right] = 0 \\
\frac{\partial E}{\partial \beta} &= \sum_{i=1}^{N} 2c_{i,y} \left[(p_i - q_i) \cdot \vec{n}_i + T \cdot \vec{n}_i + \vec{r} \cdot c_i \right] = 0 \\
\frac{\partial E}{\partial \gamma} &= \sum_{i=1}^{N} 2c_{i,z} \left[(p_i - q_i) \cdot \vec{n}_i + T \cdot \vec{n}_i + \vec{r} \cdot c_i \right] = 0 \\
\frac{\partial E}{\partial T_x} &= \sum_{i=1}^{N} 2n_{i,x} \left[(p_i - q_i) \cdot \vec{n}_i + T \cdot \vec{n}_i + \vec{r} \cdot c_i \right] = 0 \\
\frac{\partial E}{\partial T_y} &= \sum_{i=1}^{N} 2n_{i,y} \left[(p_i - q_i) \cdot \vec{n}_i + T \cdot \vec{n}_i + \vec{r} \cdot c_i \right] = 0 \\
\frac{\partial E}{\partial T_z} &= \sum_{i=1}^{N} 2n_{i,z} \left[(p_i - q_i) \cdot \vec{n}_i + T \cdot \vec{n}_i + \vec{r} \cdot c_i \right] = 0
\end{aligned}$$

In matrix form, this system of equations becomes

$$\begin{aligned}
\sum_{i=1}^{N} \begin{bmatrix}
c_{i,x}c_{i,x} & c_{i,x}c_{i,y} & c_{i,x}c_{i,z} & c_{i,x}n_{i,x} & c_{i,x}n_{i,y} & c_{i,x}n_{i,z} \\
c_{i,y}c_{i,x} & c_{i,y}c_{i,y} & c_{i,y}c_{i,z} & c_{i,y}n_{i,x} & c_{i,y}n_{i,y} & c_{i,y}n_{i,z} \\
c_{i,z}c_{i,x} & c_{i,z}c_{i,y} & c_{i,z}c_{i,z} & c_{i,z}n_{i,x} & c_{i,z}n_{i,y} & c_{i,z}n_{i,z} \\
n_{i,x}c_{i,x} & n_{i,x}c_{i,y} & n_{i,x}c_{i,z} & n_{i,x}n_{i,x} & n_{i,x}n_{i,y} & n_{i,x}n_{i,z} \\
n_{i,y}c_{i,x} & n_{i,y}c_{i,y} & n_{i,y}c_{i,z} & n_{i,y}n_{i,x} & n_{i,y}n_{i,y} & n_{i,y}n_{i,z} \\
n_{i,z}c_{i,x} & n_{i,z}c_{i,y} & n_{i,z}c_{i,z} & n_{i,z}n_{i,x} & n_{i,z}n_{i,y} & n_{i,z}n_{i,z}
\end{bmatrix}
\begin{bmatrix} \alpha \\ \beta \\ \gamma \\ T_x \\ T_y \\ T_z \end{bmatrix} \\
= -\sum_{i=1}^{N} \begin{bmatrix}
c_{i,x} (p_i - q_i) \cdot \vec{n}_i \\
c_{i,y} (p_i - q_i) \cdot \vec{n}_i \\
c_{i,z} (p_i - q_i) \cdot \vec{n}_i \\
n_{i,x} (p_i - q_i) \cdot \vec{n}_i \\
n_{i,y} (p_i - q_i) \cdot \vec{n}_i \\
n_{i,z} (p_i - q_i) \cdot \vec{n}_i
\end{bmatrix}
\end{aligned}$$

which can be solved using standard methods.

E Rotation representations

E.1 Quaternion to rotation matrix

Let p_1 and p_2 be two points, where p_2 is obtained by rotating p_1. The rotation can be done with both a quaternion, q, and with a rotation matrix, $\mathbf{R}$.

$$P_2 = qP_1q^{-1}$$

where P_1 and P_2 are the quaternion representations of the points.

$$p_2 = \mathbf{R}p_1$$

Strategy:

- Perform the quaternion multiplication
- Obtain an expression that is equivalent to a 3×3 rotation matrix

Performing the quaternion multiplication

Let the unit quaternion be given as:

$$q = q_x\mathrm{i} + q_y\mathrm{j} + q_z\mathrm{k} + q_w = [q_x, q_y, q_z, q_w]$$

Since it is a unit quaternion it follows:

$$q^{-1} = q^* = [-q_x, -q_y, -q_z, q_w]$$

The quaternion representation of the point, p_1, is given as:

$$P_1 = [x, y, z, 0]$$

First the product of q and P_1 is calculated:

$$qP_1 = [q_x, q_y, q_z, q_w] \begin{bmatrix} 0 & -z & y & -x \\ z & 0 & -x & -y \\ -y & x & 0 & -z \\ x & y & z & 0 \end{bmatrix} = \begin{bmatrix} q_yz - q_zy + q_wx \\ -q_xz + q_zx + q_wy \\ q_xy - q_yx + q_wz \\ -q_xx - q_yy - q_zz \end{bmatrix}^T$$

Then the product of qP_1 and q^*:

$$qP_1q^* = \begin{bmatrix} q_yz - q_zy + q_wx \\ -q_xz + q_zx + q_wy \\ q_xy - q_yx + q_wz \\ -q_xx - q_yy - q_zz \end{bmatrix}^T \begin{bmatrix} q_w & q_z & -q_y & q_x \\ -q_z & q_w & q_x & q_y \\ q_y & -q_x & q_w & q_z \\ -q_x & -q_y & -q_z & q_w \end{bmatrix} =$$

$$\begin{bmatrix} 2q_wq_yz - 2q_wq_zy + {q_w}^2x + 2q_zq_xz - {q_z}^2x + 2q_yq_xy - {q_y}^2x + {q_x}^2x \\ 2q_zq_yz - {q_z}^2y + 2q_zq_wx - 2q_wq_xz + {q_w}^2y - {q_x}^2y + 2q_xq_yx + {q_y}^2y \\ -{q_y}^2z + 2q_yq_zy - 2q_yq_wx - {q_x}^2z + 2q_xq_zx + 2q_xq_wy + {q_w}^2z + {q_z}^2z \\ 0 \end{bmatrix}^T$$

Obtaining a matrix

Each term in the matrix is grouped according to x, y and z.

$$p_2 = qP_1q^* =$$

$$\begin{bmatrix} x\left({q_w}^2 + {q_x}^2 - {q_y}^2 - {q_z}^2\right) + y\left(-2q_zq_w + 2q_yq_x\right) + z\left(2q_yq_w + 2q_zq_x\right) \\ x\left(2q_xq_y + 2q_wq_z\right) + y\left({q_w}^2 - {q_x}^2 + {q_y}^2 - {q_z}^2\right) + z\left(-2q_xq_w + 2q_zq_y\right) \\ x\left(-2q_wq_y + 2q_xq_z\right) + y\left(2q_wq_x + 2q_yq_z\right) + z\left({q_w}^2 - {q_x}^2 - {q_y}^2 + {q_z}^2\right) \end{bmatrix}^T$$

This can be expanded into a matrix multiplication:

$$p_2 = qP_1q^* =$$

$$\begin{bmatrix} {q_w}^2 + {q_x}^2 - {q_y}^2 - {q_z}^2 & -2q_zq_w + 2q_yq_x & 2q_yq_w + 2q_zq_x \\ 2q_xq_y + 2q_wq_z & {q_w}^2 - {q_x}^2 + {q_y}^2 - {q_z}^2 & -2q_xq_w + 2q_zq_y \\ -2q_wq_y + 2q_xq_z & 2q_wq_x + 2q_yq_z & {q_w}^2 - {q_x}^2 - {q_y}^2 + {q_z}^2 \end{bmatrix}^T \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Since the vector (x, y, z) is equivalent to the point, p_1, then the large matrix must be equivalent to a rotation matrix, $\mathbf{R}$:

$$p_2 = qP_1q^* = Rp_1$$

The matrix can be further simplified since it is a unit quaternion and thus it is given that

$$q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1$$

$$R = \begin{bmatrix} 1 - 2{q_y}^2 - 2{q_z}^2 & 2q_x q_y + 2q_w q_z & -2q_w q_y + 2q_x q_z \\ -2q_z q_w + 2q_x q_y & 1 - 2{q_x}^2 - 2{q_z}^2 & 2q_w q_x + 2q_y q_z \\ 2q_y q_w + 2q_x q_z & -2q_x q_w + 2q_y q_z & 1 - 2{q_x}^2 - 2{q_y}^2 \end{bmatrix} \tag{E.1}$$

E.2 Rotation matrix to quaternion

The relation between the unit quaternion and the rotation matrix as shown in E.1 can also be used to determine a unit quaternion from a given rotation matrix [24]. Nine equations can be extracted:

$$\begin{aligned} R_{11} &= 1 - 2{q_y}^2 - 2{q_z}^2 \\ R_{12} &= 2q_x q_y + 2q_w q_z \\ R_{13} &= -2q_w q_y + 2q_x q_z \\ R_{21} &= -2q_z q_w + 2q_x q_y \\ R_{22} &= 1 - 2{q_x}^2 - 2{q_z}^2 \\ R_{23} &= 2q_w q_x + 2q_y q_z \\ R_{31} &= 2q_y q_w + 2q_x q_z \\ R_{32} &= -2q_x q_w + 2q_y q_z \\ R_{34} &= 1 - 2{q_x}^2 - 2{q_y}^2 \end{aligned}$$

The nine equations holds only four unknowns, which gives plenty of methods for solving the problem.

The goal is to obtain a unit quaternion:

$$q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1$$

Which means that the scalar part of the quaternion q_w can be expressed as:

$$\begin{aligned} q_w^2 &= 1 - q_x^2 - q_y^2 - q_z^2 \;\Leftrightarrow \\ 4q_w^2 &= 4 - 4q_x^2 - 4q_y^2 - 4q_z^2 \;\Leftrightarrow \\ 4q_w^2 &= 1 + 1 - 2q_y^2 - 2q_z^2 + 1 - 2q_x^2 - 2q_z^2 + 1 - 2q_x^2 - 2q_y^2 \;\Leftrightarrow \\ 4q_w^2 &= 1 + R_{11} + R_{22} + R_{33} \;=\; 1 + \mathrm{tr}\,(\mathbf{R}) \;\Leftrightarrow \\ q_w &= \pm\frac{\sqrt{1 + \mathrm{tr}\,(\mathbf{R})}}{2} \end{aligned}$$

The other quaternion elements can be determined from the off-diagonal elements of the matrix:
q_x:

$$R_{23} - R_{32} = (2q_w q_x + 2q_y q_z) - (-2q_x q_w + 2q_z q_y) \;=\; 4q_x q_w \;\Leftrightarrow$$
$$q_x = \frac{R_{23} - R_{32}}{4q_w}$$

q_y:

$$R_{31} - R_{13} = (2q_y q_w + 2q_x q_z) - (-2q_w q_y + 2q_x q_z) \;=\; 4q_y q_w \;\Leftrightarrow$$
$$q_y = \frac{R_{31} - R_{13}}{4q_w}$$

q_z:

$$R_{12} - R_{21} = (2q_x q_y + 2q_w q_z) - (-2q_z q_w + 2q_x q_y) \;=\; 4q_z q_w \;\Leftrightarrow$$
$$q_z = \frac{R_{12} - R_{21}}{4q_w}$$

This method has a few numerical problems. The method is only valid if

$$\mathrm{tr}\,(\mathbf{R}) > -1$$

- Case: $\mathrm{tr}\,(\mathbf{R}) < -1$. The trace cannot be less than minus one, since that would give the square root of a negative number in the calculation of q_w, and it given that the quaternion elements must be real.
- Case: $\mathrm{tr}\,(\mathbf{R}) = -1$. The trace cannot be minus one, since that would yield $q_w = 0$ and then division by zero in the calculation of the remaining quaternion elements. The same division will also cause imprecise results should $q_w \approx 0$.

Other similar methods without these problems have been proposed [36].

E.3 Rotation matrix to axis angle

Let $\mathbf{R}$ be a 3×3 rotation matrix. From Eulers rotation theorem it is given that the eigenvalues of the matrix are $\left(1, e^{\pm \mathrm{j}\theta}\right)$ which equals:

$$\lambda = (1\ ,\ \cos\theta + \mathrm{j}\sin\theta\ ,\ \cos\theta - \mathrm{j}\sin\theta)$$

where the angle θ is the rotation angle used in the axis angle representation.

The angle

The angle can thus easily be extracted from the rotation matrix, since the trace of the matrix corresponds to the sum of the eigenvalues:

$$\mathrm{tr}\,(R) = 1 + \cos\theta + \mathrm{j}\sin\theta + \cos\theta - \mathrm{j}\sin\theta = 1 + 2\cos\theta$$

The angle can then be determined by:

$$\cos\theta = \frac{\mathrm{tr}\,(R) - 1}{2}$$

The axis:
If the rotation matrix is applied to a vector $\mathbf{u}$ that runs parallel to axis of rotation, then the result must be the vector itself:

$$\mathbf{Ru} = \mathbf{u}$$

This corresponds to the eigenvalue equation:

$$\mathbf{Ax} = \lambda\mathbf{x}$$

where λ is the eigenvalue and $\mathbf{x}$ the eigenvector.

The axis can then be determined by finding the eigenvector of $\mathbf{R}$ that has the eigenvalue of 1.

E.4 Axis angle to rotation matrix

Let $\vec{v}_{\mathrm{rot}}$ be a vector in $\mathbb{R}^3$ that is obtained by rotating the vector $\vec{v}$ an angle of θ degrees about a unit axis $\vec{z}$. The goal is to find the matrix, $\mathbf{R}$ that when multiplied on the left side of $\vec{v}$ gives $\vec{v}_{\mathrm{rot}}$:

$$\vec{v}_{\mathrm{rot}} = \mathbf{R}\vec{v}$$

Strategy:

- Project the vector $\vec{v}$ into a plane that is orthogonal to the rotation axis, thus reducing the rotation to a 2-D situation, as shown on figure E.1.
- Perform the rotation in 2-D and return to 3-D.
- Obtain an expression for a 3×3 matrix.

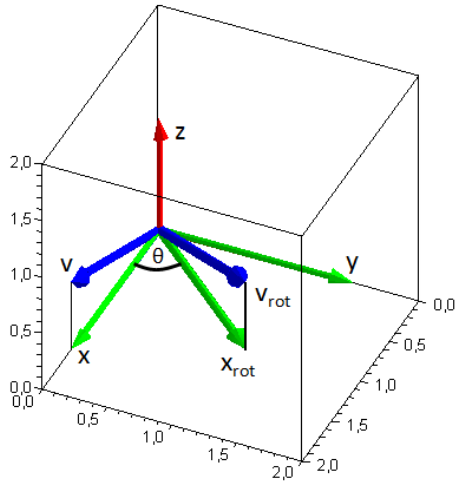


Figure E.1: Strategy for converting axis angle representation to rotation matrix. The axis of rotation $\vec{z}$ (red), the vector $\vec{v}$ before and after rotation (blue) and vectors in the plane ortogonal to $\vec{z}$ (green) are shown.

Projecting into 2-D

Let $\vec{x}$ be the projection of $\vec{v}$ into the plane orthogonal to $\vec{z}$:

$$\vec{x} = \vec{v} - \vec{z}(\vec{z} \cdot \vec{v})$$

Further let $\vec{y}$ be the vector that is orthogonal to both $\vec{z}$ and $\vec{x}$:

$$\vec{y} = \vec{z} \times \vec{v}$$

The vectors $\vec{x}$ and $\vec{y}$ have equal lengths and they span the ortogonal plane to $\vec{z}$ just like standard axes of a Cartesian coordinate system.

Rotation and returning to 3-D

Next step is to rotate $\vec{x}$ θ-degrees so that $\vec{x}_{\text{rot}}$ corresponds to the projection of $\vec{v}_{\text{rot}}$ into the 2-D plane. The rotation can be done with basic trigonometry.

$$\vec{x}_{\text{rot}} = \vec{x}\cos\theta + \vec{y}\sin\theta =$$
$$(\vec{v} - \vec{z}(\vec{z} \cdot \vec{v}))\cos\theta + (\vec{z} \times \vec{v})\sin\theta$$

The component that was subtracted from $\vec{v}$ in order to obtain $\vec{x}$, can now be added to the rotated $\vec{x}$ to obtain the rotated $\vec{v}$:

$$\begin{gathered}\vec{v}_{\text{rot}} = \vec{x}_{\text{rot}} + \vec{z}(\vec{z}\cdot\vec{v}) = \\ (\vec{v} - \vec{z}(\vec{z}\cdot\vec{v}))\cos\theta + (\vec{z}\times\vec{v})\sin\theta + \vec{z}(\vec{z}\cdot\vec{v}) = \\ \vec{v}\cos\theta + (\vec{z}\times\vec{v})\sin\theta + \vec{z}(\vec{z}\cdot\vec{v})(1-\cos\theta) \end{gathered} \tag{E.2}$$

Rotation matrix

The equation E.2 can be rewritten into matrix form:

$$\mathbf{v}_{\text{rot}} = (\mathbf{I}\cos\theta)\mathbf{v} + ([\mathbf{z}]_x \sin\theta)\mathbf{v} + (1-\cos\theta)\mathbf{z}\mathbf{z}^T\mathbf{v}$$

where $\mathbf{I}$ is a 3×3 identity matrix and $[\mathbf{z}]_x$ is the cross-product matrix of $\mathbf{z}$:

$$[\mathbf{z}]_x = \begin{bmatrix} 0 & -z_3 & z_2 \\ z_3 & 0 & -z_1 \\ -z_2 & z_1 & 0 \end{bmatrix}$$

By grouping according to $\mathbf{v}$, the following rotation matrix can be obtained:

$$\begin{gathered}\mathbf{v}_{\text{rot}} = \left(\mathbf{I}\cos\theta + [\mathbf{z}]_x\sin\theta + (1-\cos\theta)\mathbf{z}\mathbf{z}^T\right)\mathbf{v} \;\Leftrightarrow \\ \mathbf{v}_{\text{rot}} = \mathbf{R}\mathbf{v}\end{gathered}$$

The rotation matrix can further be simplified. Since $\vec{z}$ is a unit vector the following relation holds true:

$$\mathbf{z}\mathbf{z}^T = [\mathbf{z}]_x^2 + \mathbf{I}$$

Thus the rotation matrix becomes:

$$\begin{gathered}\mathbf{R} = \mathbf{I}\cos\theta + \sin\theta[\mathbf{z}]_x + (1-\cos\theta)([\mathbf{z}]_x^2 + \mathbf{I}) \;\Leftrightarrow \\ \mathbf{R} = \mathbf{I}\cos\theta + \sin\theta[\mathbf{z}]_x + [\mathbf{z}]_x^2(1-\cos\theta) + \mathbf{I} - \mathbf{I}\cos\theta \;\Leftrightarrow \\ \mathbf{R} = \mathbf{I} + \sin\theta[\mathbf{z}]_x + (1-\cos\theta)[\mathbf{z}]_x^2 \;\Leftrightarrow \\ \mathbf{R} = \mathbf{I} + \sin\theta[\mathbf{z}]_x + (1-\cos\theta)(\mathbf{z}\mathbf{z}^T - \mathbf{I})\end{gathered}$$

Bibliography

[1] J.M. Anton-Rodriguez, Mark C. Huisman, Merence Sibomana, J.C. Mathews, M. Walker, Sune H. Keller, and M.C. Asselin. Investigation of motion induced errors in scatter correction in hrrt brain scanner. Unpublished.

[2] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(5):698–700, 1987.

[3] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[4] P.J. Besl and H.D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[5] G. Blais and M.D. Levine. Registering multiview range data to create 3d computer objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):820–824, 1995.

[6] A. Bowyer. Computing dirichlet tessellations. *Computer Journal*, 24(2):162–166, 1981.

[7] Y. Chen and G. Medioni. Object modeling by registration of multiple range images. *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 2724–2729 vol.3, 1991.

[8] K. Dinelle, S. Blinder, Ju-Chieh Cheng, S. Lidstone, K. Buckley, T.J. Ruth, and V. Sossi. Investigation of subject motion encountered during a typical positron emission tomography scan. *2006 IEEE Nuclear Science Symposium Conference Record (IEEE Cat. No.06CH37832)*, page 5 pp., 2007.

[9] O.D. Faugeras and M. Hebert. The representation, recognition, and locating of 3-d objects. *International Journal of Robotics Research*, 5(3):27, 1986.

[10] Andrew W. Fitzgibbon. Robust registration of 2d and 3d point sets. *Image and Vision Computing*, 21(13-14):1145–1153, 2003.

[11] J.H. Friedman, J.L. Bentley, and R. Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.

[12] A.S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, 1984.

[13] G. Godin, M. Rioux, and R. Baribeau. Three-dimensional registration using range and intensity information. *Proceedings of the SPIE - The International Society for Optical Engineering*, 2350:279–290, 1994.

[14] M.V. Green, J. Seidel, S.D. Stein, T.E. Tedder, K.M. Kempner, C. Kertzman, and T.A. Zeffiro. Head movement in normal subjects during simulated pet brain imaging with and without head restraint. *Journal of Nuclear Medicine*, 35(9):1538–1546, 1994.

[15] Stefan Gumhold, Xinlong Wang, and Rob Macleod. Feature extraction from point clouds. In *In Proceedings of the 10 th International Meshing Roundtable*, pages 293–305, 2001.

[16] M.A. Herráez, D.R. Burton, M.J. Lalor, and M.A. Gdeisat. Fast two-dimensional phase-unwrapping algorithm based on sorting by reliability following a noncontinuous path. *Applied Optics*, 41(35):7437–7444, 2002.

[17] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics*, 26(2):71–78, 1992.

[18] B.K.P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A (Optics and Image Science)*, 4(4):629–642, 1987.

[19] Northern Digital Inc. *Polaris Vicra User Guide*, 3 edition, March 2008.

[20] Xiao Jin, T. Mulnix, B. Planeta-Wilson, J.-D. Gallezot, and R.E. Carson. Accuracy of head motion compensation for the hrrt: comparison of methods. *2009 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC 2009)*, pages 3199–3202, 2009.

[21] Timothee Jost and Heinz Hugli. A multi-resolution icp with heuristic closest point search for fast and robust 3d registration of range images. *3D Digital Imaging and Modeling, International Conference on*, 0:427, 2003.

[22] Anatoliy Kats and Mark McCartin-Lim. Point cloud alignment - accelerated with a gpu. Master's thesis, University of Toronto, 2009.

[23] D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

[24] J.B. Kuipers. *Quaternions and rotation sequences : a primer with applications to orbits, aerospace, and virtual reality.* Princeton University Press, 1999. Bog laant fra biblioteket.

[25] T. Masuda, K. Sakaue, and N. Yokoya. Registration and integration of multiple range images for 3-d model construction. *Proceedings of 13th International Conference on Pattern Recognition*, 1:879–883 vol.1, 1996.

[26] Yongwei Miao, Jieqing Feng, and Qunsheng Peng. Curvature estimation of point-sampled surfaces and its applications. *Computational Science and Its Applications - ICCSA 2005. International Conference. Proceedings, Part III (Lecture Notes in Computer Science Vol. 3482)*, pages 1023–1032, 2005.

[27] Tomas Moller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. *Doktorsavhandlingar vid Chalmers Tekniska Hogskola*, (1425):109–115, 1998.

[28] Oline Olesen, Flemming Andersen, Soeren Holm, Joergen Jensen, Sune Keller, Merence Sibomana, Claus Svarer, and Liselotte Hoejgaard. A new tool fixation for external 3d head tracking using the polaris vicra system with the hrrt pet scanner. *J NUCL MED MEETING ABSTRACTS*, 50(2 MeetingAbstracts):1528–, 2009.

[29] O.V. Olesen. Structured light 3d tracking system for measuring motions in pet brain imaging. Master's thesis, Technical University of Denmark, 2007.

[30] O.V. Olesen, M. Sibomana, S.H. Keller, F. Andersen, J. Jensen, S. Holm, C. Svarer, and L. Højgaard. Spatial resolution of the hrrt pet scanner using 3d-osem psf reconstruction. *2009 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC 2009)*, pages 3789–3790, 2009.

[31] Rasmus R. Paulsen, Jakob A. Baerentzen, and Rasmus Larsen. Markov random field surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 16, 2010.

[32] Mark Pauly, Markus Gross, and Leif P. Kobbelt. Efficient simplification of point-sampled surfaces. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 163–170, Washington, DC, USA, 2002. IEEE Computer Society.

[33] S. Rebay. Efficient unstructured mesh generation by means of delaunay triangulation and bowyer-watson algorithm. *Journal of Computational Physics*, 106(1):125–138, 1993.

[34] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, 2001.

[35] G. Schaufler and H.W. Jensen. Ray tracing point sampled geometry. *Rendering Techniques 2000. Proceedings of the Eurographics Workshop*, pages 319–417, 2000.

[36] R.A. Spurrier. Comment on "singularity-free extraction of a quaternion from a direction-cosine matrix". 1978.

[37] M M Ter-Pogossian, M E Phelps, E J Hoffman, and N A Mullani. A positron-emission transaxial tomograph for nuclear imaging (PETT). *Radiology*, 114(1):89–98, 1975.

[38] D. F. Watson. Computing the n-dimensional delaunay tessellation with application to voronoi polytopes. *Computer Journal*, 24(2):167–172, 1981.

[39] Andrew D. Wiles, David G. Thompson, Donald D. Frantz, Andrew D. Wiles, and David G. Thompson. Accuracy assessment and interpretation for optical tracking systems. *Proceedings of SPIE - The International Society for Optical Engineering*, 5367:421–432, 2004.