TECHNICAL UNIVERSITY OF DENMARK

M.SC.

Point Cloud Registration and Procedures for Optical Acquisition

Author:
Tim Felle Olsen
s134589

Supervisor:
J. Andreas Berærntzen

January 27, 2019

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Abstract

Digital 3D scanning of general objects often include manual processes. No scan can be done in one sweep since the object will always be held or rest in a way, which obscures parts of the object. This project seeks to reduce the manual part by automating a software based alignment of partially overlapping scans of an object.

The goal of the project is to describe and implement a global alignment algorithm, ultimately get the algorithm implemented as part of the DTU 3D scanning software.

The proposed algorithm is developed by Zhou et al. 2016 and boasts global efficient 3D registration without the need of a local initialisation, like the classical Iterative Closest Point algorithm do. The full mathematical background of the algorithm is explained in detail. Additionally thorough testing of the algorithm have been performed to find eventual limitations, which is an important aspect of the project. Tests are performed both on a small model of the Stanford Bunny with some modifications and large scans obtained at DTU Imaging.

The implementation performs well on small datasets where parameters can be tweaked easily, however it is not robust enough to handle real scanned datasets. The main faults of the implementation lie within the Fast Point Feature Histogram estimation and correspondence matching. The performance of the implementation can be optimised further.

The implementation by the original authors performed well in all tests and computation time were quite efficient, so with further development it is possible to integrate the method in a 3D Scanner pipeline.

Danish summary

Manuelle processer kan sjældent undgås, når generelle objekter 3D-skannes digitalt. Skanninger kan på nuværende tidspunkt ikke fange hele objekter, da objektet altid skal ligge eller holdes, hvilket obskurer dele af objektet. I dette projekt forsøges det at reducere mængden af manuelt arbejde, der skal til at afstemme orienteringen af delvis overlappende skanninger for et objekt.

Målet med projektet er at beskrive og implementere en global afstemnings algoritme, ultimativt undersøges det om en sådan algoritme kan implementeres i et realtidssystem til skanning af objekter på DTU.

Den foreslåede algoritme er udviklet af Zhou et al. 2016 er i stand til effektiv global afstemning, uden brug af et lokalt udgangspunkt som den klassiske Iterative Closest Point algoritme kræver. Projektet inkluderer en fuld gennemgang af den matematiske baggrund, der skal til for at forstå og implementere algoritmen. Endvidere er algoritmen blevet testet i et forsøg på at identificere begrænsninger og mangler. Test er blevet udført både på Stanford Kaninen og på skanninger af objekter foretaget på DTU.

Implementeringen giver gode testresultater for kanindatasættet, hvor parametre kan blive justeret let, implementeringen er ikke robust nok til at håndtere skanninger. Problemerne for implementeringen ligger primært i udregningen af Fast Point Feature Histogrammerne og korrespondanceestimeringen. Ydermere kan optimeringer på beregningstiden stadig foretages.

Den originale implementering af Zhou et al. 2016 præsterer godt i alle test og beregningstiderne er lave, derfor med videreudvikling af implementeringen er det muligt at integrere algoritmen i en 3D-skannings proces.

Acknowledgement

I would like to extend a few thanks, invaluable support have been given by my supervisor and the people present at our weekly meetings. Many great ideas and recommendations have been passed my way and the project would not have gone smoothly without it.

Additionally Marcus Krogh have been a great support, both in sanity checks of the optimisation theory in the project and as moral support at the DTU library.

The support teams of both Open3D and DTU Computing Center have been great in assisting with issues regarding the software and system they administrate. Responses have been quick and precise.

Some of the datasets used in this project was provided by the DTU Imaging section. These datasets were raw data from scanning several objects and helped greatly to examine the methods working on real sets of data.

Contents

CHAPTER 1 Introduction

Companies are specialising in methods to construct digital three dimensional models from the real world, making 3D scanners a very widespread technology. A vast variety of objects are scanned ranging from teeth to landscapes, and most of the methods used today are based on photogrammetry. Google and other companies are using photos taken from air-crafts to construct height maps of the terrain used for maps and other applications. The cinematography industry spend vast amounts of time on photogrammetry, using large camera set-ups to capture very precise 3D models of actors, these models can then be used as base material for computer graphics. Although widespread as it is, challenges still exist for the technology.

The imaging section at the Technical University of Denmark have built a 3D scanner in the form of a structured light scanner. The scanner is constructed from a turntable for the object, a projector, which projects a number of line based patterns onto the object, these patterns are used to find correspondences between the images taken by the two cameras (See Figure 1.1 for an illustration). The correspondences are then used in a triangulation to define a 3D point cloud. However, the point clouds created need to be cleaned and modified before they are useful.

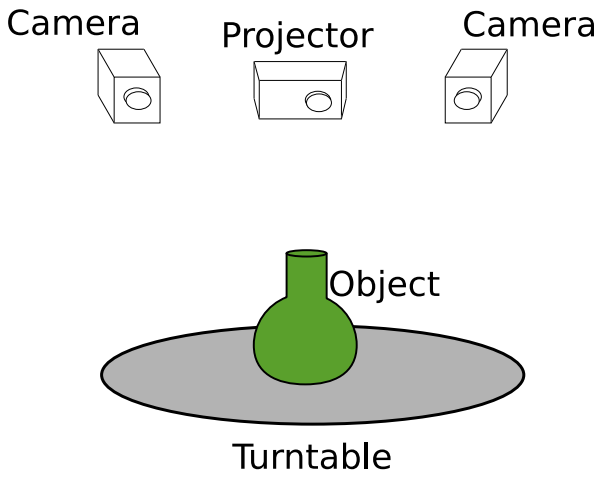


Figure 1.1: Sketch of the DTU 3D scanner.

At any given time only parts of the object can be seen by the cameras. Because of this, a number of partial scans are created using the turntable. The partial scans can be aligned since the rotation between each scan is known from the turntable.
However, the object is resting on the turntable and the contact area will not be part of any scan. To obtain a full model, the object must be manually reoriented inside the scanner. This produce a number of partial scans which must then be correctly aligned in order to complete the model of the object. This process of aligning partial 3D models to each other is called a registration process and is the goal of this project.

The project aims to describe, implement and test a global alignment method, which can be used for registration of multiple scans. The method should be automatic and robust making it possible to integrate as part of the 3D scanner pipeline currently active at DTU Imaging. Since the implementation should be part of a real-time application it must be efficient to avoid significant delays while yielding a proper level of precision.

1.1 Global and local alignment

Within the fields of computer vision and computational geometry two different issues in regards to alignment of models exist, the local and the global alignment problem. Local alignment refer to a situation where a good guess of the transformation is known, or only small changes have been made between each model. Given the turntable setup defined above, if we did not record how much the turntable changes between each scan, but we do know the change is relatively small, we could use a local method to find the transformation between the models.

However, when an object is completely reoriented on the turntable the change is in general too large for a local alignment to be possible. The local methods would probably find some local minima but the models would not be aligned properly. This is where global alignment methods are important. These methods aim to be robust toward any initial positions of the models. In general two approaches are used, try to find the transformation directly or estimate some rough transformation and let a local refinement take over when the solution is close. The direct approaches often use a feature driven or statistical method to locate correspondences and then align these.

1.1.1 Background and existing work

Estimation of rigid object transformations have been done for many years and over time the general problem size has increased. Today point clouds including millions of points are generated from scanners and the alignment of these needs to be efficient. An analysis by Eggert et al. 1997 deemed closed form solutions superior to iterative methods, both in terms of robustness and efficiency. However, all of these methods rely on computing the Singular Value Decomposition or the Eigen System of some matrices set up during the computations. These measures require some regularity in the data and are prone to errors under noise additions. Additionally these methods were efficient at the time but the increase in computational power and optimisations in the iterative methods have rendered these closed form solutions obsolete.

One of the most, if not the most, popular method for estimating alignments of data sets is the Iterative Closest Point (ICP) algorithm. ICP is a method, which have been altered and improved by many scientists since its initial development in the early 80's [Rusinkiewicz and Levoy 2001]. The method is a simple local registration, fairly efficient and will always converge to a local minima. These features have made it a popular refinement method for rough global methods. However, there do exist a number of problems with this method, as the many attempts to improve it show. First of all, even if the method is efficient, it requires a Nearest Neighbour search in every iteration, which is not an efficient computation.

In an attempt to accelerate the algorithm Qiu et al. 2009 implemented a GPU accelerated Nearest Neighbour Search. Qiu and the other authors did achieve a significant speed-up for the method compared to the original CPU based method however, their method did not tackle the algorithmic complexity of ICP, it only achieves a way to use more resources. Additionally the method still requires an initial transformation of the model.

Another optimisation to the ICP algorithm was done by Yang et al. 2013. These authors tackled the problem of local minima. The ICP algorithm is very susceptible to local minima in the optimisation, with outliers and noise it can produce some poor results. The changes made in the article by Yang, included a Branch and Bound outer loop computing an initial guess for the ICP algorithm, which then refine the intermediate result. This way Branch and Bound was used to traverse the solution space searching for a better solution and ICP ensured this solution was locally optimal. This addition to the algorithm did indeed remove the need for an initial alignment of the surfaces, but still gave no significant improvement of the algorithmic complexity.

In recent years machine learning and data mining have received a great deal of attention, and have opened up a wide range of data and feature driven methods for a wide range of mathematical disciplines. Back in 2008 a couple of scientists at the Technical University of Munich published a method to give ICP a good initial

transformation, by pairing correspondences between surfaces and estimating a transformation based on these pairs. The article by Rusu, Blodow, Marton, et al. 2008 describes the features they estimated and how this gave a good initial transformation, even for big sets of data without relying on statistical methods like the Fuzzy method by Tarel and Boujemaa 1995 does. This feature based method handles the initial alignment of surfaces quite efficiently and performs quite well. Improvements to the performance of the method and removal of redundant features was done in the article by Rusu, Blodow, and Beetz 2009. However, their estimation of the transformation relied on a sub-sampling method of the points matched by the Fast Point Feature Histograms (FPFH) developed in the article.

Zhou et al. 2016 used the FPFH to estimate point correspondences between several surfaces and then defined a minimisation problem, which can be solved directly to find a global solution to the registration problem. This algorithm was called Fast Global Registration and is the main focus of this project. Zhou and the other authors show great results and efficient calculations compared to the ICP based methods primarily since Nearest Neighbour search is no longer needed in every iteration. But the article describing the method contains a number of steps not described in full, thus it is difficult to implement the method based on the article.

CHAPTER 2 Theoretical setup

In the article "Fast Global Registration" by Zhou et al. 2016, an algorithm under the same name boasts efficient global alignment of partially overlapping surfaces, without the need for preprocessing an initial alignment. The algorithm is based on correspondences from feature matching between the surfaces and an optimisation problem including a robust penalisation. The goal of this chapter is to give a thorough explanation of the algorithm and theories behind it. Some parts of the article are not well described, primarily the connection to Lie Groups and definition of residuals and Jacobians for the optimisation problem. These parts will be explained in detail here.

2.1 Mathematical background

Rigid transformations on $\mathbb{R}^n$ are a subset of the affine transformations. Affine transformations are defined through $y = Ax + t$, but when A is a rotation matrix t can be interpreted as a translation vector. These transformations can be represented by a $(n+1) \times (n+1)$ matrix [Eade 2017]. The introduction of an additional dimension for the matrix is the result of homogeneous coordinates playing a part [Aanæs 2018].

Homogeneous coordinates are a representation of points in $\mathbb{R}^n$ through a vector in $\mathbb{R}^{n+1}$. Representation in homogeneous coordinates add a scaling factor as an additional dimension to the vector. This scaling is also applied to the other dimensions by multiplication $x = (x_1, x_2, ..., x_n) = (sx_1, sx_2, ..., sx_n, s) = x_{hom}$. The mapping between inhomogeneous and homogeneous are done by finding the vector such that this scaling is 1. This mean a point can be represented as homogeneous coordinates by an infinite amount of vectors. $(4, 6, 2)$, $(2, 3, 1)$, $(1, 1.5, 0.5)$ all represent the point $(2, 3)$ in 2D.

Homogeneous coordinates are used widely in computer-vision and graphics. A light source in a scene, infinitely far away can be represented with a direction by the homogeneous vector $(4, 3, 1, 0)$ since the mapping back to inhomogeneous 3D coordinates would lead to the point, $(4/0, 3/0, 1/0) \approx (\infty, \infty, \infty)$ which is at infinity but does not retain information about the direction of the light.

The benefit of using homogeneous coordinates for our problem becomes apparent with the affine transformations. For a regular point an affine transformation is described by the product of a rotation matrix and the addition with a translation vector. This operation does however simplify into a linear matrix multiplication in homogeneous coordinates,

$$\begin{pmatrix} y \\ 1 \end{pmatrix} = T \begin{pmatrix} x \\ 1 \end{pmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ 1 \end{pmatrix} = \begin{pmatrix} Rx + t \\ 1 \end{pmatrix}.$$

This formulation of points in 3D connects the rotation and translation to the transformation matrix T. These types of transformations represented as matrices and vectors are one of the primary reasons to use homogeneous coordinates in computer-vision.

The matrix T is an element of the Lie Group called $SE(n)$. $SE(n)$ is called the group of *direct affine isometries* or, more intuitively for the subject of this thesis, the group of *rigid motions* in n dimensions.
Simplifying the transformation to a linear map rather than an affine one is not the only benefit of this formulation. The Lie Groups are topological groups as well as manifolds [Gallier 2001], meaning tangent spaces and other favourable properties exists for these groups.

For the optimisation procedure, described in section 2.3, differentials of the transformation matrices are needed. This is where the introduction of Lie Algebras are needed. Since the $SE(3)$ group is a semi-direct product of $SO(3)$ and $\mathbb{R}^3$ we can use some of their properties to simplify the derivation of the Lie Algebra called $\mathfrak{se}(3)$.

From Gallier 2001 we know that the tangent basis for rotation matrices $SO(3)$ is the set of skew symmetric

matrices. Finding the differential directions for the translation part of $SE(3)$ is quite simple. From this we can define the elements of the algebra by the following basis elements,

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

These 6 matrices represent the directional derivatives of the tangent space for $SE(3)$. The vector $\xi = (\alpha, \beta, \gamma, a, b, c)$ describe the contributions of the 6 directions, 3 contributions alter the rotation and 3 alter the translation. The combination of the vector and the 6 matrices define the Lie Algebra $\mathfrak{se}(3)$,

$$\Omega = \begin{bmatrix} 0 & -\gamma & \beta & a \\ \gamma & 0 & -\alpha & b \\ -\beta & \alpha & 0 & c \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

In order to prove that this is the algebra associated with $SE(3)$ we need to examine the curve $\omega(t) = \exp(t\Omega)$, where the matrix exponential is defined as the series [Hartley 2016],

$$\exp(A) = I + \sum_{n=1}^{\infty} \frac{A^n}{n!}.$$

If $\omega(t) \in SE(3),\ \forall t \in \mathbb{R}$, then Ω define the Lie Algebra. First split the Lie Algebra as follows,

$$\Omega = \begin{bmatrix} U & u \\ 0 & 0 \end{bmatrix}.$$

Now examine the curve for any time t,

$$\begin{aligned} \omega(t) &= \exp(t\Omega) \\ &= I + \sum_{n=1}^{\infty} \frac{(t\Omega)^n}{n!} \\ &= I + \sum_{n=1}^{\infty} \frac{t^n}{n!}\Omega^n \\ &= I + \frac{t}{1!}\begin{bmatrix} U & u \\ 0 & 0 \end{bmatrix} + \frac{t^2}{2!}\begin{bmatrix} U^2 & Uu \\ 0 & 0 \end{bmatrix} + \frac{t^3}{3!}\begin{bmatrix} U^3 & U^2u \\ 0 & 0 \end{bmatrix} + \cdots \\ &= \begin{bmatrix} I + \frac{t}{1!}U + \frac{t^2}{2!}U^2 + \frac{t^3}{3!}U^3 + \cdots & \frac{t}{1!}u + \frac{t^2}{2!}Uu + \frac{t^3}{3!}U^2u + \cdots \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \exp(tU) & Vu \\ 0 & 1 \end{bmatrix}, \\ V &= I + \frac{t^2}{2!}U + \frac{t^3}{3!}U^2 + \cdots. \end{aligned}$$

The multiplication tU does not change that U is skew symmetric, therefore we know from Hartley 2016 that $\exp(tU) \in SO(3)$ is a rotation matrix. So we now have to show V is not divergent.

First let us examine the effect of multiplying a skew symmetric matrix with itself.

$$U = \begin{bmatrix} 0 & -\gamma & \beta \\ \gamma & 0 & -\alpha \\ -\beta & \alpha & 0 \end{bmatrix}.$$

$$\begin{aligned} U^2 &= \begin{bmatrix} 0 & -\gamma & \beta \\ \gamma & 0 & -\alpha \\ -\beta & \alpha & 0 \end{bmatrix} \begin{bmatrix} 0 & -\gamma & \beta \\ \gamma & 0 & -\alpha \\ -\beta & \alpha & 0 \end{bmatrix} \\ &= \begin{bmatrix} -\gamma^2 - \beta^2 & \alpha\beta & \gamma\alpha \\ \alpha\beta & -\gamma^2 - \alpha^2 & \gamma\beta \\ \alpha\gamma & \beta\gamma & -\beta^2 - \alpha^2 \end{bmatrix}. \end{aligned}$$

$$\begin{aligned} U^3 &= \begin{bmatrix} -\gamma^2 - \beta^2 & \alpha\beta & \gamma\alpha \\ \alpha\beta & -\gamma^2 - \alpha^2 & \gamma\beta \\ \alpha\gamma & \beta\gamma & -\beta^2 - \alpha^2 \end{bmatrix} \begin{bmatrix} 0 & -\gamma & \beta \\ \gamma & 0 & -\alpha \\ -\beta & \alpha & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & -\left(-\beta^2 - \gamma^2\right)\gamma + \gamma\alpha^2 & \left(-\beta^2 - \gamma^2\right)\beta - \beta\alpha^2 \\ \left(-\alpha^2 - \gamma^2\right)\gamma - \gamma\beta^2 & 0 & \beta^2\alpha - \left(-\alpha^2 - \gamma^2\right)\alpha \\ \gamma^2\beta - \left(-\alpha^2 - \beta^2\right)\beta & -\gamma^2\alpha + \left(-\alpha^2 - \beta^2\right)\alpha & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & \left(-\alpha^2 - \beta^2 - \gamma^2\right)(-\gamma) & \left(-\alpha^2 - \beta^2 - \gamma^2\right)\beta \\ \left(-\alpha^2 - \beta^2 - \gamma^2\right)\gamma & 0 & \left(-\alpha^2 - \beta^2 - \gamma^2\right)(-\alpha) \\ \left(-\alpha^2 - \beta^2 - \gamma^2\right)(-\beta) & \left(-\alpha^2 - \beta^2 - \gamma^2\right)\alpha & 0 \end{bmatrix} \\ &= \left(-\alpha^2 - \beta^2 - \gamma^2\right) U. \end{aligned}$$

So after 3 multiplications a pattern appear. This pattern can be exploited for proving V is not diverging, lets describe the rotational part of ξ by the symbol $\xi_R = (\alpha, \beta, \gamma)$. The factor in front of the matrix can then be described as $-(\xi_R^\top \xi_R)$.

But since this is used in an infinite sum, examine how the matrices can be expressed for an arbitrary index [Eade 2017].

$$\begin{aligned} \theta^2 &= \xi_R^\top \xi_R, \\ U^{2n+1} &= (-1)^n \theta^{2n} U, \\ U^{2n+2} &= (-1)^n \theta^{2n} U^2. \end{aligned} \tag{2.1}$$

With these observations we are ready to examine V again. The benefit of the above is, that one might rewrite the sum to find a known infinite series. It is possible to split the sum into even and odd indices and use Equation 2.1.

$$
\begin{aligned}
V &= I + \sum_{n=1}^{\infty} \frac{t^{n+1}}{(n+1)!} U^n \\
&= I + \sum_{n=0}^{\infty} \frac{t^{n+2}}{(n+2)!} U^{n+1} \\
&= I + \sum_{n=0}^{\infty} \frac{t^{2n+2}}{(2n+2)!} U^{2n+1} + \sum_{n=0}^{\infty} \frac{t^{2n+3}}{(2n+3)!} U^{2n+2} \\
&= I + \sum_{n=0}^{\infty} \frac{t^{2n+2}}{(2n+2)!} \left((-1)^n \theta^{2n} U \right) + \sum_{n=0}^{\infty} \frac{t^{2n+3}}{(2n+3)!} \left((-1)^n \theta^{2n} U^2 \right) \\
&= I + t^2 U \sum_{n=0}^{\infty} (-1)^n \frac{(t\theta)^{2n}}{(2n+2)!} + t^3 U^2 \sum_{n=0}^{\infty} (-1)^n \frac{(t\theta)^{2n}}{(2n+3)!} \\
&= I + \frac{t^2}{(t\theta)^2} U \sum_{n=0}^{\infty} (-1)^n \frac{(t\theta)^{2n+2}}{(2n+2)!} + \frac{t^3}{(t\theta)^3} U^2 \sum_{n=0}^{\infty} (-1)^n \frac{(t\theta)^{2n+3}}{(2n+3)!}
\end{aligned}
$$

From Christensen 2012 we know that these sums are very close to the cosine and sine series. Adjust for the differences and we can describe the sums by the following,

$$
\begin{aligned}
\sum_{n=0}^{\infty} (-1)^n \frac{(t\theta)^{2n+2}}{(2n+2)!} &= 1 - \cos(t\theta), \\
\sum_{n=0}^{\infty} (-1)^n \frac{(t\theta)^{2n+3}}{(2n+3)!} &= t\theta - \sin(t\theta).
\end{aligned}
$$

Introducing these into V clear the rest of the way to a solution.

$$
\begin{aligned}
V &= I + \frac{t^2}{(t\theta)^2} U \sum_{n=0}^{\infty} (-1)^n \frac{(t\theta)^{2n+2}}{(2n+2)!} + \frac{t^3}{(t\theta)^3} U^2 \sum_{n=0}^{\infty} (-1)^n \frac{(t\theta)^{2n+3}}{(2n+3)!} \\
&= I + \frac{t^2}{(t\theta)^2} U (1 - \cos(t\theta)) + \frac{t^3}{(t\theta)^3} U^2 (t\theta - \sin(t\theta)) \\
&= I + \frac{1 - \cos(t\theta)}{\theta^2} U + \frac{t\theta - \sin(t\theta)}{\theta^3} U^2.
\end{aligned}
$$

This prove V is convergent for any value of t since $||\xi_R|| \neq 0$ for any skew symmetric matrix.

Since V is convergent $Vu \in \mathbb{R}^3$ for any t and $\omega(t) = \exp(t\Omega) \in SE(3),\ \forall t \in \mathbb{R}$, so Ω is the Lie Algebra for $SE(3)$.

2.2 Correspondences

Finding good correspondences is paramount for finding a good registration, since these are the points used for minimising the distance between the surfaces. Most iterative algorithms re-estimates the correspondences in each iteration, which is often a big performance hit. In this algorithm the correspondences are estimated initially and then they are not updated again. Since the correspondences are only computed once then a robust system must be used for estimating these. The correspondences are estimated through the "Fast Point Feature Histograms" developed by Rusu, Blodow, and Beetz 2009 because of the great robustness.

Establishing the correspondences is done in a couple of stages, first the feature histograms are computed for all points on each surface. Features are computed on a number of different scales to determine points holding

unique features. Second step is to match up points on different surfaces which hold similar features. Correspondences are tested to reduce the amount of incorrect pairs as the final step.

The entire process from feature estimation to correspondence pairing is summarised in Algorithm 3.1.

2.2.1 Feature estimation

For all points (p) in the data set the goal is to estimate a set of features, which represent a local patch of the surface. The surface normal (n) is used for all points so if these are not present from the dataset they must be estimated. Features are estimated by finding the neighbourhood $\mathcal{N}_k$ which contain all points with a distance smaller than r_k to p. For each $p_k \in \mathcal{N}_k$ assign p and p_k an index i or j, these indices are given to make the feature consistent regardless of the order of p and p_k. The index i is assigned to the point with the smallest angle between the surface normal at the point and the line connecting the points $\angle(n_i, p-p_k) < \angle(n_j, p-p_k)$, j is then assigned to the other point. Now a Darboux frame is defined by the 3 vectors,

$$u = n_i, \qquad v = (p_j - p_i) \times u, \qquad w = u \times v.$$

The features are computed as angles relating the Darboux frame to the normals of the two points,

$$\begin{aligned} f_1 &= v \cdot n_j, \\ f_2 &= \frac{u \cdot (p_j - p_i)}{||p_j - p_i||}, \\ f_3 &= \arctan\left(w \cdot n_j, u \cdot n_j\right). \end{aligned}$$

Here $\arctan(a, b)$ is a robust implementation of $\tan^{-1}\left(\frac{a}{b}\right)$.

These features are then organised into a 6 bin histogram by checking if the value of the feature is above or below a certain threshold (s_1, s_2 or s_3). These histograms are summed up over all neighbours and divided by the number of neighbours. This way the feature histogram, which Rusu, Blodow, and Beetz 2009 called the Simplified Point Feature Histogram (SPFH), describe the percentage of points in the neighbourhood, which have features falling in each category.

In order to better describe the neighbourhood of each point a second step is computed. Once SPFH has been determined for all points, Fast Point Feature Histograms (FPFH) can be computed by the following computation. Given a point p go through all neighbours and add a weighted contribution to the SPFH for p,

$$\mathcal{F}(p) = \mathcal{F}_s(p) + \frac{1}{K} \sum_{p_k \in \mathcal{N}_k} \frac{1}{||p - p_k||} \mathcal{F}_s(p_k),$$

with K being the number of neighbours.

2.2.2 Uniqueness and persistence of features

The features estimated above describe the surface around each point, but they do not describe if that point is unique in any way. A plane will have the same features for all points on the interior of it, however the corners and edges will have different features. In order to determine the unique points on the surface a bit of statistics is applied. First the average histogram μ is computed, then all points are ranked after the signed distance to the average histogram for each feature. These distributions can be modelled by Gaussian distributions and can then be used to define a cut off for unique points far away from the mean,

$$||\mathcal{F}(p) - \mu|| > \alpha\sigma,$$

where α is a scale defining how unique the point should be and σ is the standard deviation of the estimated Gaussian. The set of unique points for the neighbourhood size r_k is called P_k.

To ensure the chosen features are valid over different scales of the neighbourhood, all of the above is computed for a multiple of radii r_k. Then the set of points used to describe the surface will be the points that are marked as unique for all the chosen values of r_k,

$$\mathcal{P} = \bigcap P_k.$$

2.2.3 Defining the correspondences

After computing the features and determining the persistent points for the surfaces in question it is now possible to pair them up to define the set of correspondences. A Nearest Neighbour search is conducted between the surfaces in feature space to list the closest neighbour of $\mathcal{P}_1$ and $\mathcal{P}_2$. These correspondences will be refined by two additional operations. A reciprocity test is done, meaning only points which are mutually nearest neighbours will be classified as a correspondence. Lastly a tuple test is performed by picking 3 random correspondences from the dataset [Zhou et al. 2016]. The 3 correspondence pairs are marked as compatible if the ratio of the distances between the points in each set is close to 1,

$$\forall i \neq j, \quad \tau < \frac{||p_i - p_j||}{||q_i - q_j||} \leq \frac{1}{\tau},$$

where $\tau = 0.9$. The computation here ensures the distances between points on surface 1 and surface 2 is roughly the same for all correspondences. If points a and b are paired with respectively c and d, then the pairs, (a, c) and (b, d), are only valid if the distance between a and b is about the same as c and d. Using 3 pairs give a robust set of correspondences.

2.3 Optimisation

The correspondences can be used to define a minimisation problem used to estimate the transformation between surfaces. The optimisation problem is defined by Zhou et al. 2016, and is here replicated with parts of the derivation expanded upon to clarify how the algorithm should be implemented. Algorithm 3.2 shows a short version of the theory presented in this section.

2.3.1 Defining the objective

The objective function defined by Zhou et al. 2016 is defined by the following equation,

$$\arg\min_{T} \sum_{(p,q)\in\mathcal{K}} \rho(||p - Tq||).$$

The goal is to estimate the 4×4 transformation matrix T. p and q are represented using homogeneous coordinates since the transformation can be represented as a linear function instead of an affine one. The penalty function (ρ) used by Zhou and the others is a scaled Geman-McClure estimator, where the penalty scale is called ν,

$$\rho(x) = \frac{\nu x^2}{\nu + x^2}.$$

Additionally a line process ($\mathbb{L} = \{l_{p,q}\}$) is introduced in order to handle discontinuities in the objective[Black and Rangarajan 1996]. Introducing this line process changes the objective of the minimisation to the following.

$$E(T, \mathbb{L}) = \sum_{(p,q)\in\mathcal{K}} l_{p,q} \left|\left|p - Tq\right|\right|^2 + \sum_{(p,q)\in\mathcal{K}} \Psi(l_{p,q}), \tag{2.2}$$

with a prior term Ψ defined by,

$$\Psi(l_{p,q}) = \nu \left(\sqrt{l_{p,q}} - 1\right)^2.$$

This introduction must not change the optimal solution, therefore an optimal solution with respect to the line process must also be an optimal solution to the original problem. To show that the solution is preserved examine the partial derivative of Equation 2.2 with respect to $l_{p,q}$,

$$\frac{\partial E}{\partial l_{p,q}} = \left|\left|p - Tq\right|\right|^2 + \nu \frac{\sqrt{l_{p,q}} - 1}{\sqrt{l_{p,q}}}.$$

This can be used to find $l_{p,q}$ for a solution which corresponds to the partial derivative vanishing,

$$\begin{aligned}
0 &= \left|\left|p - Tq\right|\right|^2 + \nu \frac{\sqrt{l_{p,q}} - 1}{\sqrt{l_{p,q}}}, \\
0 &= \left|\left|p - Tq\right|\right|^2 + \nu - \nu \frac{1}{\sqrt{l_{p,q}}}, \\
\frac{1}{\sqrt{l_{p,q}}} &= \frac{\left|\left|p - Tq\right|\right|^2 + \nu}{\nu}, \\
\sqrt{l_{p,q}} &= \frac{\nu}{\left|\left|p - Tq\right|\right|^2 + \nu}, \\
l_{p,q} &= \left(\frac{\nu}{\nu + \left|\left|p - Tq\right|\right|^2}\right)^2.
\end{aligned}$$

Reintroducing this result into the original objective gives the following,

$$
\begin{aligned}
E(T,\mathbb{L}) &= \sum_{(p,q)\in\mathcal{K}} l_{p,q}\left|\left|p-Tq\right|\right|^2 + \sum_{(p,q)\in\mathcal{K}} \Psi(l_{p,q}) \\
&= \sum_{(p,q)\in\mathcal{K}} \left(\frac{\nu}{\nu+\left|\left|p-Tq\right|\right|^2}\right)^2 \left|\left|p-Tq\right|\right|^2 + \sum_{(p,q)\in\mathcal{K}} \Psi(l_{p,q}) \\
&= \sum_{(p,q)\in\mathcal{K}} \left(\frac{\nu\left|\left|p-Tq\right|\right|}{\nu+\left|\left|p-Tq\right|\right|^2}\right)^2 + \sum_{(p,q)\in\mathcal{K}} \nu\left(\sqrt{\left(\frac{\nu}{\nu+\left|\left|p-Tq\right|\right|^2}\right)^2} - 1\right)^2 \\
&= \sum_{(p,q)\in\mathcal{K}} \left(\frac{\nu\left|\left|p-Tq\right|\right|}{\nu+\left|\left|p-Tq\right|\right|^2}\right)^2 + \sum_{(p,q)\in\mathcal{K}} \nu\left(\frac{\nu}{\nu+\left|\left|p-Tq\right|\right|^2} - 1\right)^2 \\
&= \sum_{(p,q)\in\mathcal{K}} \left(\left(\frac{\nu\left|\left|p-Tq\right|\right|}{\nu+\left|\left|p-Tq\right|\right|^2}\right)^2 + \left(\frac{\sqrt{\nu}\nu}{\nu+\left|\left|p-Tq\right|\right|^2} - \sqrt{\nu}\right)^2\right) \\
&= \sum_{(p,q)\in\mathcal{K}} \left(\frac{\nu^2\left|\left|p-Tq\right|\right|^2}{\left(\nu+\left|\left|p-Tq\right|\right|^2\right)^2} + \frac{\left(\sqrt{\nu}\nu - \sqrt{\nu}(\nu+\left|\left|p-Tq\right|\right|^2)\right)^2}{\left(\nu+\left|\left|p-Tq\right|\right|^2\right)^2}\right) \\
&= \sum_{(p,q)\in\mathcal{K}} \frac{\nu^2\left|\left|p-Tq\right|\right|^2 + \left(\sqrt{\nu}\nu - \sqrt{\nu}\nu + \sqrt{\nu}\left|\left|p-Tq\right|\right|^2\right)^2}{\left(\nu+\left|\left|p-Tq\right|\right|^2\right)^2} \\
&= \sum_{(p,q)\in\mathcal{K}} \frac{\nu^2\left|\left|p-Tq\right|\right|^2 + \nu\left|\left|p-Tq\right|\right|^4}{\left(\nu+\left|\left|p-Tq\right|\right|^2\right)^2} \\
&= \sum_{(p,q)\in\mathcal{K}} \frac{\nu\left|\left|p-Tq\right|\right|^2\left(\nu+\left|\left|p-Tq\right|\right|^2\right)}{\left(\nu+\left|\left|p-Tq\right|\right|^2\right)^2} \\
&= \sum_{(p,q)\in\mathcal{K}} \frac{\nu\left|\left|p-Tq\right|\right|^2}{\nu+\left|\left|p-Tq\right|\right|^2} \\
&= \sum_{(p,q)\in\mathcal{K}} \rho(\left|\left|p-Tq\right|\right|).
\end{aligned}
$$

This derivation verifies that introducing and optimising over the line process does not change the solution to the problem.

2.3.2 Solving the optimisation problem

The objective function defined in Equation 2.2 contains two parameters, however an alternating method called Coordinate Descent Method [Nocedal and Wright 1999, Chapter 9] can be employed to solve the problem. The method freezes one parameter and solves a step for the other, alternating between which variable is frozen and which is being optimised.

When T is fixed the solution for $\mathbb{L}$ can be determined. This was done already during the derivation of the partial derivative of the objective. So when freezing T and updating $\mathbb{L}$, the solution is to update each $l_{p,q}$ by,

$$
l_{p,q} = \left(\frac{\nu}{\nu+\left|\left|p-Tq\right|\right|^2}\right)^2.
$$

To update T fix $\mathbb{L}$. Having $\mathbb{L}$ fixed leads to a couple of simplifications to the objective, firstly the prior term Ψ can be removed and secondly the problem can be formulated as a non-linear least squares problem. These types of problems have a residual based objective on the following form,

$$E(T) = \frac{1}{2} ||e(T)||^2,$$

where $e(T)$ is the residual. This is a well known type of problem and several methods have been developed to solve this. Here the method called Gauss-Newton will be used [Nocedal and Wright 1999, Section 10.3]. In order to do so the Lie Algebra is now introduced describing a step in the tangent plane for the transformation, however we use a linear approximation,

$$\begin{aligned} T^{k+1} &= \exp(\Omega) T \\ &\approx \begin{bmatrix} 1 & -\gamma^k & \beta^k & a^k \\ \gamma^k & 1 & -\alpha^k & b^k \\ -\beta^k & \alpha^k & 1 & c^k \\ 0 & 0 & 0 & 1 \end{bmatrix} T^k \\ &= \Xi^k T^k. \end{aligned}$$

The Gauss-Newton method is a iterative scheme where the variable is updated in each step through solving the linear system,

$$J_e^\top J_e \xi = -J_e^\top e(T).$$

So next step is to compute the residual $e(T)$ and the Jacobian of the residual J_e with respect to the variable ξ. A very important note here is to map the matrix Ω back into the space $SE(3)$ before updating T. This is done through the matrix exponential.

With the objective defined we can determine the residual vector in the k'th iteration,

$$E(T) = \sum_{(p,q)\in\mathcal{K}} l_{p,q} \left|\left|p - \Xi^{k-1} T^{k-1} q\right|\right|^2.$$

Since each $l_{p,q}$ is a positive number, they can be moved inside the norm. The sum of squared norms can also be reformulated by stacking the interior of the norm into a $4K$ vector, K being the number of correspondences. Here the k index is suppressed to assist readability,

$$\begin{aligned} E(T) &= ||e(T)||^2, \\ e_i(T) &= \sqrt{l_{p_i,q_i}}(p_i - \Xi T q_i). \end{aligned}$$

So every e_i consists of a 4D vector. The Jacobian of this can be computed and will end up as a 4×6 matrix for each e_i.

$$J_e = \frac{\partial}{\partial \xi} \left(\sqrt{l_{p_i,q_i}}(p_i - \Xi T q_i) \right).$$

The product of Tq does not contain any information about ξ, so define a vector $M = Tq$ which reduce the size

of the equations.

$$
\begin{aligned}
J_e &= \frac{\partial}{\partial \xi}\left(\sqrt{l_{p_i,q_i}}\,(p_i - \Xi M)\right) \\
&= \frac{\partial}{\partial \xi}\left(\sqrt{l_{p_i,q_i}}\left(p_i - \begin{bmatrix} 1 & -\gamma & \beta & a \\ \gamma & 1 & -\alpha & b \\ -\beta & \alpha & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{bmatrix}\right)\right) \\
&= \frac{\partial}{\partial \xi}\left(\sqrt{l_{p_i,q_i}}\left(p_i - \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{bmatrix} - \begin{bmatrix} -\gamma m_2 + \beta m_3 + a m_4 \\ \gamma m_1 - \alpha m_3 + b m_4 \\ -\beta m_1 + \alpha m_2 + c m_4 \\ 0 \end{bmatrix}\right)\right) \\
&= \frac{\partial}{\partial \xi}\left(\sqrt{l_{p_i,q_i}}\left(p_i - \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{bmatrix} - \begin{bmatrix} 0 & m_3 & -m_2 & m_4 & 0 & 0 \\ -m_3 & 0 & m_1 & 0 & m_4 & 0 \\ m_2 & -m_1 & 0 & 0 & 0 & m_4 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} \alpha \\ \beta \\ \gamma \\ a \\ b \\ c \end{bmatrix}\right)\right) \\
&= \frac{\partial}{\partial \xi}\left(\sqrt{l_{p_i,q_i}} p_i - \sqrt{l_{p_i,q_i}}\begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{bmatrix} - \sqrt{l_{p_i,q_i}}\begin{bmatrix} 0 & m_3 & -m_2 & m_4 & 0 & 0 \\ -m_3 & 0 & m_1 & 0 & m_4 & 0 \\ m_2 & -m_1 & 0 & 0 & 0 & m_4 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}\xi\right) \\
&= \sqrt{l_{p_i,q_i}}\begin{bmatrix} 0 & -m_3 & m_2 & -m_4 & 0 & 0 \\ m_3 & 0 & -m_1 & 0 & -m_4 & 0 \\ -m_2 & m_1 & 0 & 0 & 0 & -m_4 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.
\end{aligned}
$$

All of these derivations give the final expressions for the stacked residual and Jacobian,

$$
\begin{aligned}
e(T) &= \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_K \end{bmatrix}, \quad \text{where } e_i = \sqrt{l_{p_i,q_i}}(p_i - \Xi T q_i). \\
J_e(T) &= \begin{bmatrix} J_{e_1} \\ J_{e_2} \\ \vdots \\ J_{e_K} \end{bmatrix}, \quad \text{where } J_{e_i} = \sqrt{l_{p_i,q_i}}\begin{bmatrix} 0 & -m_3 & m_2 & -m_4 & 0 & 0 \\ m_3 & 0 & -m_1 & 0 & -m_4 & 0 \\ -m_2 & m_1 & 0 & 0 & 0 & -m_4 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \qquad (2.3) \\
M &= \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{bmatrix} = T q_i.
\end{aligned}
$$

The residuals and the Jacobian have now been defined and can be used in the implementation.

CHAPTER 3 Implementations

All implementations are written in the language C++ and set up in a Microsoft Visual Studio (2017) solution. Makefiles have been written primarily to run on the DTU server, these are available as well. Additionally a mix of shell and matlab programming are used for testing and visualising the data and results. All code and data used in the implementation and testing of the program is included in the github repository `https://github.com/timfelle/PointCloudRegistration` and marked with version `v1.0`.

PointCloudRegistration	*Folder*	Folder containing the implementation.
/lib	*Sub folder*	Header files.
/projects	*Sub folder*	Visual Studio Projects.
/src	*Sub folder*	Source files.
/PointCloudRegistration.sln	*.sln file*	Visual Studio Solution for the project.
Testing	*Folder*	Folder containing testing scripts.
/data	*Sub folder*	Data used in the tests.
/matlab	*Sub folder*	Matlab code used in testing and visualising.
/shell	*Sub folder*	Shell scripts defining tests of the system.
/submit	*Sub folder*	Files containing settings for LSF submission.
/runtests.sh	*.sh file*	Shell script designed to run tests.
/submit.sh	*.sh file*	Shell script for submitting tests to the DTU server.
/clean_tests.sh	*.sh file*	Shell script designed to clean test folders.

Table 3.1: Folder structure of the Github repository.

3.1 General functions and information

In Table 3.1 an outline of the repository structure is shown. The structure is chosen to give a clear separation of testing and implementation. This chapter will describe the implementation in depth. First describing the general structure, including an additional program designed to generate datasets used for testing. Emphasis will be made on the implementations of the FPFH based correspondence estimation as described in section 2.2 and the actual registration algorithm described in section 2.3.

The algorithms require a number of advanced data structures and functions. So we use two libraries to ensure good implementations of the essential backbone of the program. Open3D is a library designed to handle a wide range of geometric and computer vision tasks [Zhou et al. 2018]. Eigen3 is needed by Open3D to run along with a couple of other dependencies, but the vector and matrix operations defined by Eigen is used directly as well [Guennebaud, Jacob, et al. 2010].

3.1.1 GenerateData

In order to test the project, consistent ways of generating test data must be established. Therefore the program `GenerateData` was written. The program accepts the name of a point cloud, saved in a `.ply` file, and the desired file name for the output. The program will then modify the point cloud based on a couple of environment variables set by the user. The full documentation is located in the `README_gen` file on github.

The program has a couple of functionalities. Firstly, it can transform a given model by a translation vector and a pitch-yaw-roll vector as the rotation. These vectors are specified as environment variables for the program, which are specified by `ROTATION=pitch,yaw,roll` and `TRANSLATION=x,y,z`. Rotations are doubles measured in radians and the translations are any double. The default values of these are 0 to make sure the point cloud is not changed unintentionally.

In addition to transforming the model, noise can be added in two different ways. Gaussian noise can be added to all points in the dataset or outliers can be added by moving a randomly selected subset of the data with a large Gaussian noise. The noise addition is controlled through the environment variables `NOISE_TYPE`, `NOISE_STRENGTH` and `OUTLIER_AMOUNT`. Noise strength refer to the standard deviation for the Gaussian noise and amount of outliers is measured as how large a percentage of points in the model should become outliers.

Addition of noise is done through the following expressions,

$$\begin{aligned} &\text{Gaussian: } \tilde{p} = p + \delta, && p \in \mathcal{S},\ \delta \sim \mathcal{N}(0, 1\% \cdot R \cdot strength). \\ &\text{Outliers: } \tilde{p}_i = p_i + \delta, && p_i \sim \mathcal{U}(\mathcal{S}),\ \delta \sim \mathcal{N}(0, 10\% \cdot R). \end{aligned}$$

Here $\mathcal{N}$ and $\mathcal{U}$ represents the normal and uniform distributions. Outlier indices are pulled from a uniform distribution and the noise additions are Gaussians with different standard deviations. The standard deviations are dependent on the radius (R) of the model and scales relatively well for different models.

In Figure 3.1 the Stanford Bunny [Stanford Computer Graphics Laboratory 1994] has been subjected to each of the capabilities of the data generation program. It is clear to see that the program works as intended. The test script `generateData.sh` was used to set-up these figures.

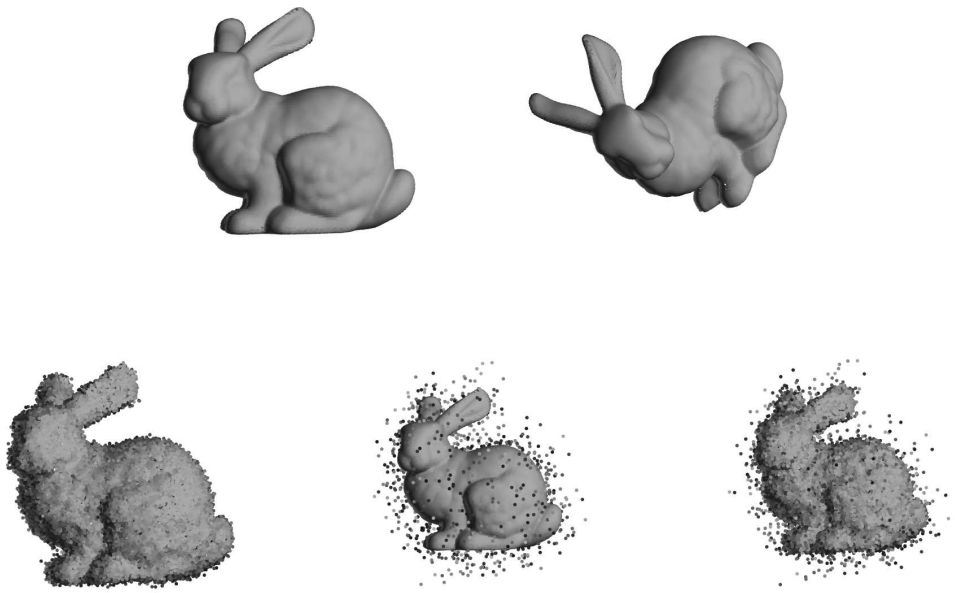


Figure 3.1: These figures show the effects of using the `GenerateData` program on a Stanford bunny model. From the top left is a clean version, a version rotated $30, 30, 45$ measured in degrees, Gaussian noise with strength parameter set to 1.5, 5% outliers added and a version with both Gaussian and outliers added.

Variable Name	Default Value	Description
INPUT_PATH	../Testing/data/	Location of data.
OUTPUT_PATH	../Testing/logs/debugging/	Location of transformed models.
ROTATION	$0.0, 0.0, 0.0$	Model rotation.
TRANSLATION	$0.0, 0.0, 0.0$	Model translation.
NOISE_TYPE	none	Type of noise added.
NOISE_STRENGTH	0.0	Strength of Gaussian noise.
OUTLIER_AMOUNT	0.0	Amount of outliers generated.

Table 3.2: Overview of environment variables for the `GenerateData` program.

3.1.2 Registration

The program `Registration` is the main driver for the surface registration. This program will handle inputs, outputs and environment variables used to control the functionality. The full documentation for this driver is also present in the `README` located in the PointCloudRegistration folder.

The environment variables, a short explanation and the default values are listed in Table 3.3.

Variable Name	Default Value	Description
INPUT_PATH	../Testing/data/	Location of data.
OUTPUT_PATH	../Testing/logs/debugging/	Location of transformed models.
OUTPUT_NAME	result	Base name for the output files.
EXPORT_CORRESPONDENCES	false	Should correspondences be exported.
FPFH_VERSION	project	Project or open3d library version.
FGR_VERSION	project	Project or open3d library version.
TOL_NU	10^{-6}	See section 3.3.
TOL_E	10^{-6}	See section 3.3.
INI_R	0.0001	See section 3.2.
END_R	0.0100	See section 3.2.
NUM_R	100	See section 3.2.
ALPHA	1.50	See section 3.2.

Table 3.3: Overview of environment variables for the `Registration` program.

3.2 Correspondence estimation

In order to implement the registration algorithm, correspondences for the system to align must be computed first. This is done through the function `computeCorrespondances`, which accepts a vector of point clouds as input and returns a list of correspondence indices. Implementation is split in two stages, first estimate features and find persistent points for each surface, then find correspondence pairs between the surfaces and attempt to eliminate erroneous pairs.

In section 2.2 the theoretical background of the FPFH algorithm is described and Algorithm 3.1 show a pseudo code outline of the algorithm. However the kd-tree data structure, as described in subsection 3.2.2, is an important part of this algorithm.

In this algorithm neighbours are computed several times so a kd-tree is built on the point cloud positions as part of the initialisation. Then for each scale (r_k) the neighbours of each point is found through the `searchRadius` function. The neighbourhood will always include the point we search around since its part of the point cloud, therefore it must be removed from the neighbourhood list.

Features are computed and thresholded into the Simplified Point Feature Histograms for each point in the neighbourhood list. The Fast Point Feature Histograms are collected and stored in a matrix structure from the eigen library. All contributions are added together and the number of scales which contain persistent points are counted. This way the FPFH is averaged across the scales for all points.

The averaging of features across scales are done in order to help matching between surfaces since the last computation of FPFH is not done on the same scale for all surfaces. Again the kd-tree structure is useful. The feature matrix is cleaned for all non-persistent points and organised into a kd-tree. This kd-tree is then passed to the `nearestNeighbour` function. The function locates the persistent points on surface 2 which have features similar to the persistent points on surface 1. The function is used in both directions so two lists of possible correspondences are set up.

Now the correspondences are cleaned first by a mutual test, ensuring the nearest neighbours are only saved if the relation is mutual. Secondly the tuple test described in the end of section 2.2 is carried out. As part of the tuple test points are randomly selected. Randomly selecting points might examine 2 valid pairs with an invalid pair, then none of the pairs will be marked valid, therefore the process is repeated several times.

In order to control the computation a couple of environment variables are used. The initial and final scale along with the number of scales are defined through the three variables `INI_R`, `END_R` and `NUM_R`. The cut-off for when a point is persistent is controlled through the variable `ALPHA`, so a point is persistent if,

$$\text{dist}(\mathcal{F}(p), \mu) > \alpha\sigma,$$

where μ is the average feature histogram and σ is the standard deviation for the distance distribution.

These statistic measures only make sense if the scale chosen is capturing geometry with distinct features, so the selection of scales needs to be done carefully.

3.2.1 Scale selection

An important part of the correspondence computation is selecting scales. The scales referred to here are the sizes of neighbourhoods used to compute the features of each point. Selecting the scales in a poor way may result in bad or non-existing correspondences so it is crucial these are selected carefully.

In the implementation scales are selected through the ends of the scale interval `INI_R` and `END_R` which represent a fraction of the bounding box radius, for example from 1% (`INI_R = 0.01`) to 5% (`END_R = 0.05`) of the radius. Then the number of scale jumps are read and (`NUM_R`) equidistant scales are used. This way the computation time can be kept fairly low. Using the bounding box radius as a guide to sizes allow models measured in different units to be used without knowing if they are measured in meters or millimetres.

In the computation the two endpoints can be interchanged, computing from big to small scales or from small to big. One direction might be better suited to reduce computation time. On some models, especially with a lot of small Gaussian noise, a lot of points will be marked persistent in small scales while very few will be persistent for large scales. Here starting by computing the big scales might give a slight performance increase, but since points must be persistent across all scales the result will be the same.

Another more robust way of selecting the scales could be to compute the features across all scales for all points. This way some analysis could be done for the points, allowing a couple of scales to be non-persistent for a point while still allowing it to be marked persistent. This could help avoid problems with bad scale range selection. At the moment it is up to the user to find a good range.

3.2.2 Kd-trees

K-Dimensional Balanced Search Trees (kd-trees [De Berg et al. 2008, Chapter 5]) is a data structure often used to handle searching in multidimensional space. The idea behind the trees are the same as for a regular Balanced Search Tree, we build a data structure which simplifies lookup at the cost of building the structure. Our multi-dimensional space will be split along each dimension in turn by the median. This way all data is structured into the search tree and Nearest Neighbour Searching will be easier. The data structure is especially important for setting up correspondences between the 6-dimensional feature spaces.

Constructing a kd-tree rely on a geometric approach to database queries. The database "space" is split into areas with half the datapoints on each side in every branching of the tree. This way it is possible to search through and find elements in the tree in $\mathcal{O}(\sqrt{n}+k)$ time with k denoting the number of points reported [De Berg et al. 2008]. With a construction time of $\mathcal{O}(n\log(n))$ the kd-tree is well worth considering when large amounts of small queries are done through a large database. Finding neighbours in a small neighbourhood around points in a 3D model fits that description quite well.

Algorithm 3.1 Correspondences using Fast Point Feature Histograms.

Requires: Point cloud with normals.
Initialise: Assign all points of the surface ($\mathcal{S}$) to $\mathcal{P}$. Define the set of used radii $\mathcal{R}$, define threshold for the features.

Determine FPFH for each surface:

for each radius $r_k \in \mathcal{R}$, **do**
  for each point $p \in \mathcal{S}$, **do**
    Determine the set of neighbours $\mathcal{N}_k(p)$ with a distance smaller than r_k to p.
    for each neighbour $p_k \in \mathcal{N}_k(p)$, **do**
      Order the pair (p, p_k) by angles between connecting line $l = p - p_k$ and the normals n, n_k.
      if $\angle(l, n) < \angle(l, n_k)$ **then**,
        $p_i = p$ and $p_j = p_k$,
      else
        $p_i = p_k$ and $p_j = p$.
      end if
      Set up a Darboux frame for the points p_i and p_j.

$$u = n_i, \qquad v = (p_j - p_i) \times u, \qquad w = u \times v.$$

      Compute the Simplified Point Features,

$$f_1 = v \cdot n_j, \qquad f_2 = \frac{u \cdot (p_j - p_i)}{||p_j - p_i||}, \qquad f_3 = \arctan\left(w \cdot n_j, u \cdot n_j\right).$$

    end for – each p_k
    Set up the Simplified Point Feature Histograms $\mathcal{F}_s$ with the thresholds s_1, s_2 and s_3.
  end for – each p
  Compute Fast Point Feature Histogram for each $p \in \mathcal{S}$,

$$\mathcal{F}(p) = \mathcal{F}_s(p) + \frac{1}{K} \sum_{p_k \in \mathcal{N}_k(p)} \frac{1}{||p - p_k||} \mathcal{F}_s(p_k).$$

  Compute the mean histogram,

$$\mu = \frac{1}{P} \sum_{p \in \mathcal{S}} \mathcal{F}(p),$$

  Compute distance to the mean histogram and standard deviation of distances.
  If the distance between $\mathcal{F}(p)$ and μ is less than $\alpha \cdot \sigma$ then remove p from $\mathcal{P}$.
end for – each r_k

Set up the correspondence set:

Compute $\mathcal{K}_I$, by nearest neighbour search between $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$ in FPFH space.
Compute $\mathcal{K}_{II}$, by reciprocity test.
Must be mutual closest neighbour.
Compute $\mathcal{K}_{III}$, by tuple test.
Pick 3 random correspondence pairs then the following must hold,

$$\forall i \neq j, \quad \tau < \frac{||p_i - p_j||}{||q_i - q_j||} < \frac{1}{\tau},$$

where $\tau = 0.9$.
Assign $\mathcal{K} = \mathcal{K}_{III}$.

3.3 Pairwise Fast Global Registration

The pairwise FGR algorithm is implemented in the function `fastGlobalRegistrationPair` which accepts inputs for the set of correspondence indices and two point cloud datasets. The pseudo code algorithm is shown in Algorithm 3.2.

The implementation follow the algorithm described in Algorithm 3.2 closely, but it is worth noting the matrix exponential. This is a function not under active support from the Eigen library authors, which make inclusion of `unsupported/Eigen/MatrixFunctions` necessary. This matrix exponential will map our Lie Algebra element Ω back into $SE(3)$ as described in section 2.1, so it is of vital importance to have computed.

Environment variables used by this function is the tolerances on penalisation (ν) through `TOL_NU` and on the average error used in convergence checks $(\text{mean}_{p,q}(||p - Tq||),\ (p,q) \in \mathcal{K})$ by `TOL_E`.

Algorithm 3.2 Pairwise Fast Global Registration [Zhou et al. 2016]

Initialise: tolerance δ and diameter D of the largest surface, and compute correspondences between the surfaces through Algorithm 3.1.

1: Initialise $T = I$, $\xi = 0$ and $\nu = D^2$.
2: **while** Not converged or $\nu > \delta^2$ **do**
3: Set $J_e = 0$, $e = 0$.
4: **for** each $(p, q) \in \mathcal{K}$, **do**
5: Compute $l_{(p,q)}$,

$$l_{(p,q)} = \left(\frac{\nu}{\nu + ||p - T^k q||^2} \right)^2 .$$

6: Compute $M = T^k q$.
7: Update J_e and e,

$$e_i = \sqrt{l_{p_i,q_i}}(p_i - T^k q_i), \qquad J_{e_i} = \sqrt{l_{p_i,q_i}} \begin{bmatrix} 0 & -m_3 & m_2 & -m_4 & 0 & 0 \\ m_3 & 0 & -m_1 & 0 & -m_4 & 0 \\ -m_2 & m_1 & 0 & 0 & 0 & -m_4 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

8: **end for**
9: Update T,

$$\xi = (\alpha, \beta, \gamma, a, b, c) = -(J_e^\top J_e)^{-1} J_e^\top e, \qquad \Xi(\xi) = \exp\left(\begin{bmatrix} 0 & -\gamma & \beta & a \\ \gamma & 0 & -\alpha & b \\ -\beta & \alpha & 0 & c \\ 0 & 0 & 0 & 0 \end{bmatrix} \right),$$

$$T^{k+1} = \Xi(\xi) T^k.$$

10: Every 4 iterations, $\nu = \frac{\nu}{2}$.
11: **end while**
12: Verify whether T aligns Q to P.

CHAPTER 4 Tests and results

Streamlining and reproducibility of tests are important, so all tests used in this project are provided through Github. Additionally the test driver `runtest.sh` was written. `runtest.sh` is a simple script, which creates a temporary directory for each test, copies over the executables from the project folder and executes the tests specified through terminal inputs. This way tests will not interfere with each other even if run in parallel or updated in the `shell` folder during execution.

The performance tests are run on the DTU cluster. The CPU used for all computations are the Intel Xeon CPU E5-2650 v4 2.20GHz see Appendix B for more info on the CPU used.

The simplest test done is just to see if the algorithm works. The bunny have been copied to two separate parts and a part of the model have been deleted from each. One of the models are transformed and the goal is to realign the parts. Figure 4.1 illustrates the result of the registration program. The result is very pleasing since the models are aligned almost perfectly. However this situation can hardly be used to evaluate the performance of the model, this is a clean model and a 3D scanner will introduce a range of noise and imperfections. So more testing is required to evaluate the performance of the program.

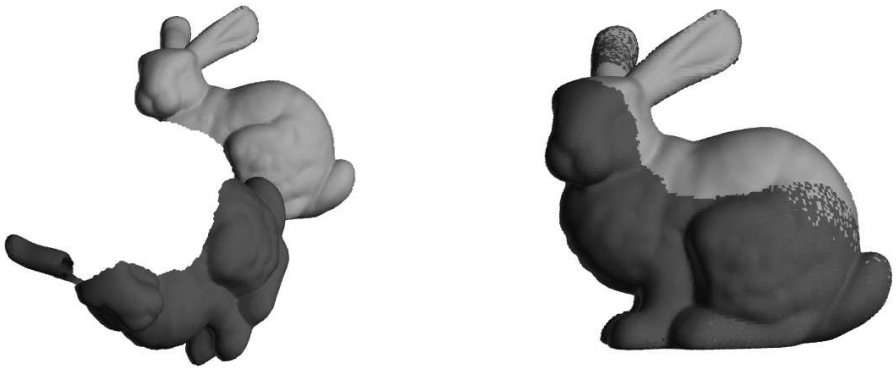


Figure 4.1: Figure showing the clean bunny before and after running the registration.

4.1 Synthetics

In order to analyse the performance of our method we start by generating synthetic data. This synthetic data will be based on the Stanford bunny [Stanford Computer Graphics Laboratory 1994]. The bunny will be subjected to a number of deformations using the `GenerateData` program described in subsection 3.1.1. The tests will be done both for outliers, Gaussian noise and a mixture of the two. For all tests the initial position of the partial bunnies are as shown in Figure 4.1 then the data generation program is used to alter the point cloud. All synthetic tests can be replicated using the test script `noiseTest`.

The original transformation is printed from the data generation program, which is one of the good reasons to generate the synthetic data. In order to determine how well the registration is done we look at the difference between the original transformation and the estimated one. And to quantify the difference the Frobenius norm is computed,

$$Err = ||T_{true} - T_{est}||_F \,.$$

4.1.1 Outliers

First of all 3D scanners often introduce a large number of outlier points. Dust particles, edges on the background and specular reflections are some of the common reasons for outliers in a 3D scanner. In the implementation outliers are simulated by scattering a percentage of the original data.

Type	Amount	Error
Clean:		$3.82\ 10^{-4}$
Outliers:	1%	$2.70\ 10^{-4}$
Outliers:	5%	$3.86\ 10^{-4}$
Outliers:	10%	$4.10\ 10^{-4}$

Table 4.1: Overview of test precision.

Three tests are set up to examine the effect of outliers. The introduced outliers represent of 1%, 5% and 10% of the point clouds. Figure 4.2 show the final registration for each of these 3 tests. The visual alignment is on point for all 3 tests, even though for the 10% test, outliers are obstructing the view.

Figure 4.2: Registration of the bunny with outliers added. The amount of outliers in each example is 1%, 5% and 10%. It is clear to see the registration is working for the two first but the amount of outliers make it hard to see the bunny in the last.

The registration program was run without changing any parameters, meaning no alteration needs to be done even with quite heavy amounts of outliers. Table 4.1 shows the normed distance to the original transformation, so the implementation is quite robust against outliers.

4.1.2 Gaussian noise

Gaussian noise addition is quite simple, however it seems to be difficult for the system to determine persistent points when Gaussian noise have been added. Figure 4.3 shows the results of two different amounts of noise. First normally distributed noise with mean 0 and standard deviation 0.01% of the bounding box radius was added. Introducing such small noise hardly give any visual alteration of the model. It is still possible to achieve a good registration. Increasing the noise to a standard deviation of 0.05% it was impossible to obtain a reasonable result at all. This clearly show that the current implementation of the feature estimation is not robust enough.

In an attempt to illustrate the effect causing the issues for our method, a couple of additional tests were done through `gaussianTest.sh`. The correspondences were visualised for a couple of large scales in order to determine if the Gaussian noise would cause features to behave differently even for large scales. We know small scales will be affected a lot by the noise. Figure 4.4 shows the correspondence points for both surfaces, the top line is a scale of 0.1 and the bottom is 0.15. There are only minor differences with and without the addition of Gaussian noise but it does not seem to be enough to cause issues.

Attempting to start at a larger scale was tested as part of the parameter tuning for the Gaussian bunny, however results were poor. This points to small scale features playing an important part in the registration process, the larger scales might not be affected by the Gaussian but they are not descriptive enough to be used on their own.

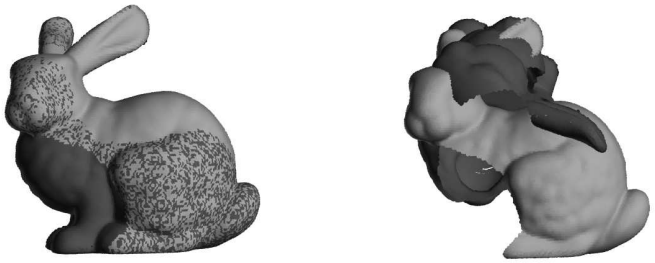


Figure 4.3: Registration of the bunny with added Gaussian noise. Addition of noise with the strength parameter set to first 0.01 then 0.05, corresponding to a standard deviation of 0.01% and 0.05% of the model

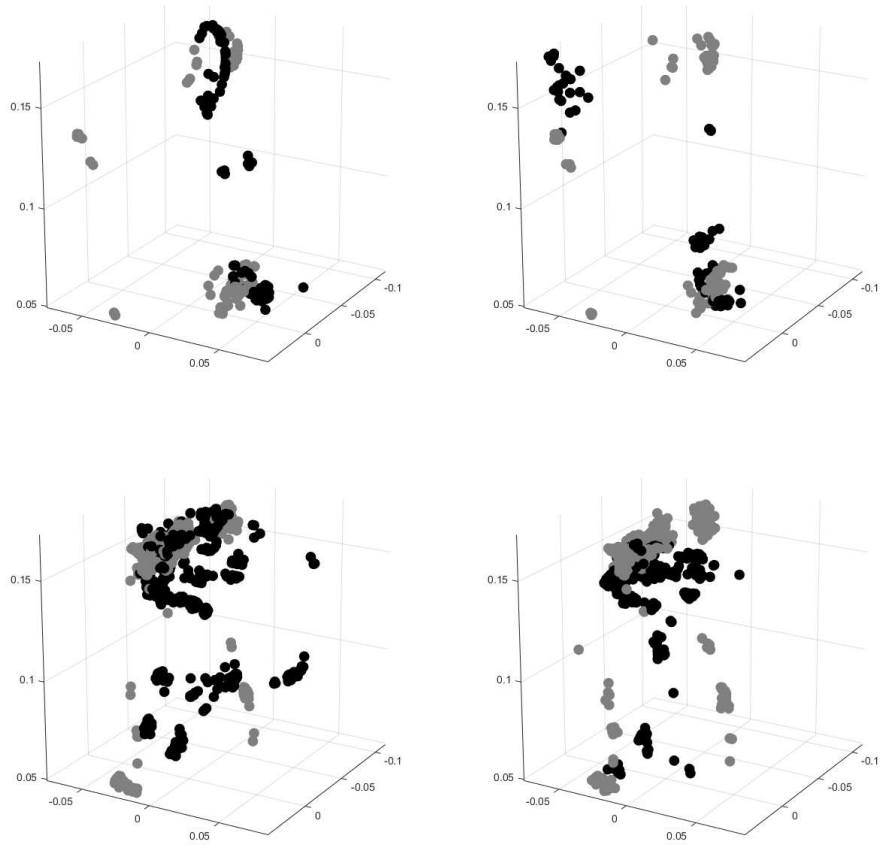


Figure 4.4: Correspondences for the clean model (left) and with Gaussian noise added (right). Top line is with a scale of 0.1 and bottom row is with a scale of 0.15.

4.1.3 Additional models

All of the above shows that the implementation works for the Stanford Bunny which is one of the classical shapes. In addition to the bunny, tests were performed on a couple of other models. The armadillo monster [CadNav 2019a] is a low-poly model, which is clear from the spacing between points. A lying giraffe [CadNav

2019b] have been separated in 2 and a bike [CadNav 2019c] have been split in 3 parts. The giraffe have a fairly uniform distribution of points, while the points on the bike is collected closely around high curvature areas like the wheels.

Both the bike and giraffe needed no alteration of parameters to complete, and the armadillo did need a bit of tweaking. However, the armadillo is small enough to pose no problem.

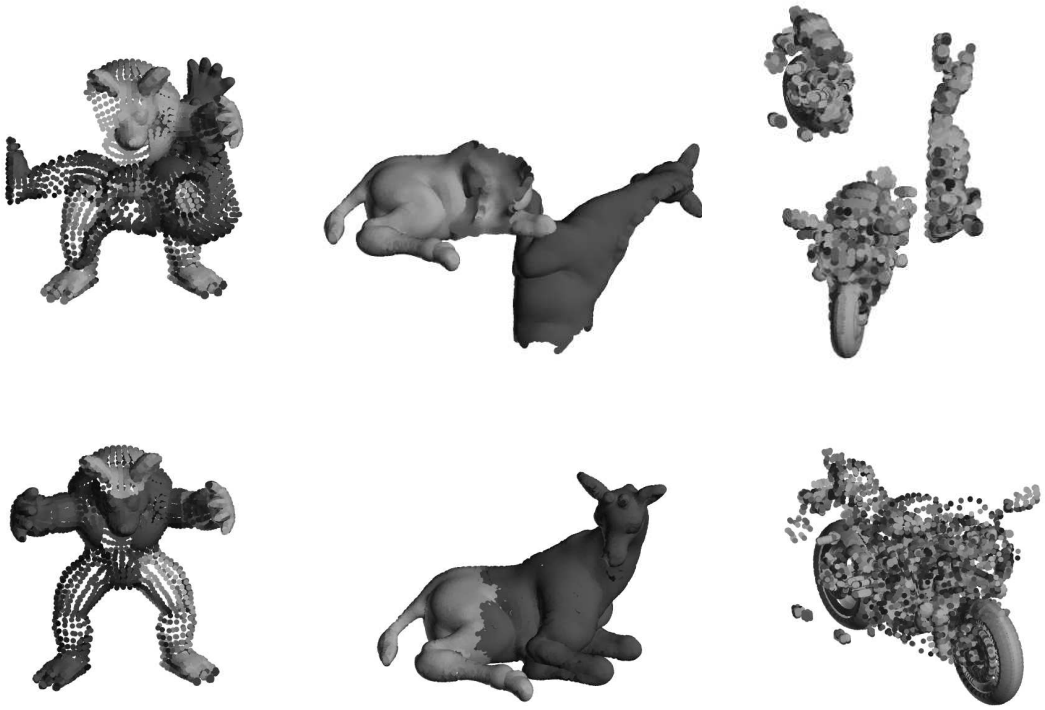


Figure 4.5: A couple of registrations for a subsampled armadillo monster, giraffe and bike.

4.2 Open3D vs project implementation

In addition to the implemented version of the algorithms Open3D included a version of these as well. These have been used to test how well the implementation is performing. A couple of things were measured, how well the different versions registered the clean bunny, along with the time it took to complete the computations. Additionally some large datasets were used to see how the implementations handled an increasing number of points and surfaces.

Version	Error	Time bunny	Time 2	Time 4	Time 6
Project	$4.27\ 10^{-5}$	14.09	1990	NA	NA
Open3D FPFH	$9.04\ 10^{-5}$	3.34	483	1679	4574
Open3D FGR	$1.68\ 10^{-6}$	3.24	20	63	137

Table 4.2: Overview of the comparison test results. Error measure and computation time for the full scale datasets are shown. Time is measured in seconds.

The project version is the implementation described in chapter 3, the Open3D FPFH version use the library version of the feature computations and the registration algorithm described in this thesis. The Open3D FGR version uses the library version to complete every step of the registration process.

The test contains 2 different parts. First an estimation of the precision for the 3 methods when the Stanford

Bunny is the model. Again parts of the bunny have been removed in the two datasets. The goal is to estimate the correct transformation and the error is measured as the Frobenious norm of the difference between the correct and the estimated transformation $\varepsilon = ||T_{true} - T_{est}||_F$. In addition to testing the precision the computation time is measured for the process of aligning the bunny.

The second part of the test is a performance test. A seal skull have been scanned using the scanner here at DTU [M. T. Olsen et al. 2016]. This dataset contain a lot of points and 18 point clouds have been generated for each orientation of the skull in the scanner. To compare the implementations the registration of 2, 4 and 6 of these point clouds have been done and the computation time is measured.

It is very clear that the project implementation is the least efficient by far. Additionally it took a long time tuning the parameters used to get decent results, however for the bunny set the precision of the methods are fairly even, with the Open3D registration yielding the best result.

For large datasets the project version is completely pointless, even just two large surfaces take ages to compute. The two other versions perform way better with the full implementation in Open3D outperforming everything as expected.

4.3 Real test

In the real world, spurious outliers come from dust, reflections or other unintended features the scanner picks up. Imperfections in the camera, or surface can cause small Gaussian like perturbations of the data points. This added to the generally large size of real data can post a range of issues for registration algorithms. The outliers and noise can make it difficult to identify good points for correspondences and make the registration less precise. The large datasets can make tuning parameters take increasingly more time since each test will take a considerable time to complete.

Two datasets have been used in testing the implementation, a scanned seal skull by M. T. Olsen et al. 2016 and a fox skull scanned by Wilm n.d. Both of the skulls have been scanned here at DTU by the scanner mentioned in the introduction. Both of these sets represent the quality expected from a real world dataset.

From the results in section 4.2 it is clear that the project implementation of FPFH features and correspondence mapping is infeasibly slow for any type of real data set. Therefore the tests here are performed by using Open3D to match correspondences and the project optimisation algorithm. In addition the full FGR implementation in Open3d have been used in order to compare the results.

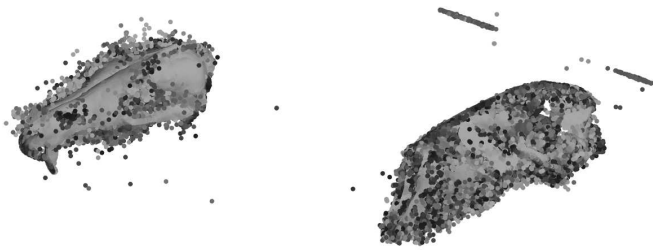


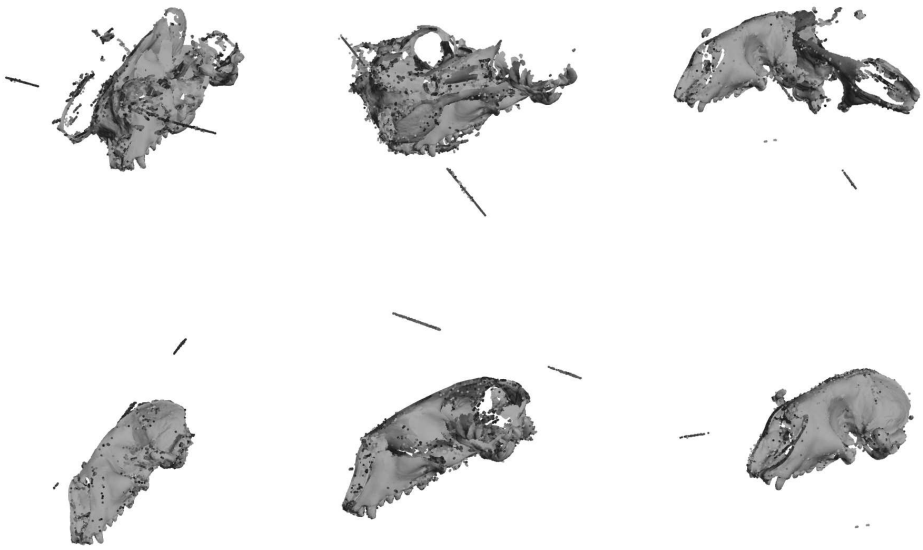
Figure 4.6: An illustration of the amount of outliers present in the scanned data.

Due to the amount of noise present within the model a small addition to the matlab routines have been added. A flag can be set in order to run an outlier removal function `pcdenoise`. Figure 4.6 clearly shows why it is needed to run such a function. The outliers make it difficult to evaluate the result. It should be noted that the registration is completed including all outliers and de-noising is only done to give a visual aid for the illustrations here.

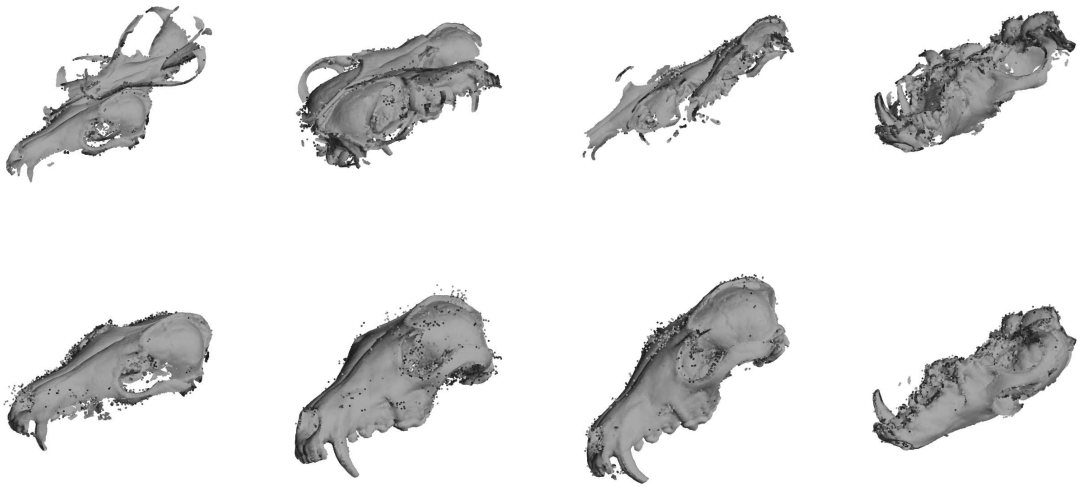
The final registration of both datasets is represented in Figure 4.7. The top 6 figures are the seal skull while the bottom 8 are the fox. The project and Open3D implementation results are shown as the top and bottom row respectively. Both methods perform fairly well on most of the sections, however a couple of the 17 point clouds do get misplaced by the project implementation. The Open3D implementation performed remarkably well and required little tweaking.

Computation time is still remarkably high for these tests, in excess of 12 hours for each dataset. This is the time spent for the complete set. And it is quite clear from section 4.2 that the Open3D implementation does not require most of this computation time.

Figure 4.7: These figures show the test results from full scale tests.



(a) Test of the seal dataset. Top row is the project implementation, bottom row contain the open3d version. The datasets are for both rows the left, right and upright.



(b) Test of the fox dataset. Top row is the project implementation, bottom row contain the open3d version. The datasets are for both rows the left, right, upright and upside down.

CHAPTER 5 Points of improvements and future work

It is clear from the testing performed, that several parts of the implementation can be improved. The goal of the project is to insert the implementation as part of a 3D scanner pipeline, however it is not nearly efficient enough to be implemented in a real-time application. So currently two major parts of the implementation should be optimised. First of all, the features are not robust enough, it requires a large amount of tweaking to get results that are even meaningful, and for large datasets this might not even be feasible due to the computation time required for each test. The second problem is computational performance. It takes way too long to get a result and for big sets of data it is infeasible to get a result.

5.1 Feature estimation

The feature estimation process seems to be the bottleneck at the moment. The implementation of the feature estimation is way slower than the Open3D version. This is of course expected, since Open3D is a well structured library, and undoubtedly performance have been an important aspect of the development. The performance of the implementation here is way slower than expected. In the comparison test it performed 5-7 times slower than the version using Open3D-FPFH. In addition to the performance, spurious outliers for the large datasets and the problems shown when adding Gaussian noise show a lack of robustness for the implementation.

A couple of things can be done to improve performance. Using some advanced techniques like openMP to run the computations multi-threaded could be done. Computing the FPFH features for all surfaces can be run in parallel which might speed-up the computations. Re-examining the inner parts of the computations and how variables are handled could possibly give some performance increase, resizing vectors or similar for each radius might be done in a better way.

Robustness of the features might be improved in several ways. One option would be to look into the statistics used to select persistent features. These statistics could be computed across all scales and one could implement a threshold to allow a point to be non-persistent for a few scales and still be marked persistent.

Looking into the bins set up for the features is another possible improvement. Currently all 3 measures are split into 2 bins. One could introduce additional bins for the measures to help separate points. Open3D seem to use a total of 36 bins, so it is possible they did some changes to how they compute the features.

Since kd-trees are already built for each surface, it is fairly efficient to search through and find points which does not have many neighbours, this could possibly be used to find outliers and neglect these from the feature computation. This could assist both performance and robustness of the correspondences.

Feature estimation could be improved by using other methods. Khoury et al. 2017 propose a method using Convolutional Neural Networks to compute a family of parametrised features. These features can then be identified on a surface very quickly giving very robust and efficient features.

5.2 Registration

The registration algorithm seems to be working quite well. It is precise and efficient, however the way multi-surface registration is done could use some additional work. Currently only pairs of surfaces are registered together, surface 0 and 1, then 1 and 2 and so on. Zhou et al. 2016 describe a method to combine the optimisations of multiple surfaces into a single computation. This included organising correspondence sets into an array and run the optimisation for an array of transformations.

An intermediate update which might improve robustness could be to merge surfaces as registrations are completed. This way a larger and larger portion of the surface is present to be aligned to and the amount of good correspondences should go up.

An idea could be to examine rings in the correspondences, meaning require correspondences to form a ring between the surfaces. Having 3 surfaces and given correspondence pairs (p_1, p_2) on surface 1 and 2, and (p_2, p_3) on surface 2 and 3, then (p_3, p_1) should also be a correspondence. This could assist the robustness of the method and could be implemented either as a hard requirement or given a weight to make the optimisation prioritise these rings of correspondences higher than regular pairs.

5.3 Alternative methods

Some of the methods mentioned in subsection 1.1.1 might have been performing well against the method described here. It would be interesting to see how much tuning is required if good results should be obtained by a method like the Go-ICP by Yang et al. 2013 or the SAC-IA used by Rusu, Blodow, and Beetz 2009 in their test of the FPFH features. The method presented in this paper did perform fairly well in the tests conducted here. Tuning parameters ended up taking a very long time and so far, satisfactory results were not found for the real datasets. The methods Go-ICP and SAC-IA show good results in the articles presenting them, just like the FGR. So a head-to-head test of the methods could be appropriate to find strengths and weaknesses for the methods.

Following the ideas proposed by Qiu et al. 2009 and Park et al. 2010, one might get a computational benefit by using the CUDA language for computations on GPU. Today the possible benefit of using GPU for computations is huge, as shown in T. Olsen and Lorenz 2018. It does require a very parallel process and quite large datasets to justify. Computing the feature estimations could provide huge speed-ups from this approach.

Extending the method to operate on triangular mesh could be a good way to allow more detailed features to be computed. Normal estimations are faster to compute and more precise, neighbours are fast to find for all points and do not require a kd-tree or similar data structure to compute. A very well established set of operations like sub-sampling are also defined for a triangular mesh. A clever sub-sampling could help neglect points on planes, planar points very rarely contain any features useful for setting up correspondences.

CHAPTER 6 Conclusion

The Fast Global Registration algorithm presented by Zhou et al. 2016 boasts high precision at a very low computational cost. Their article describe the algorithm well, even though parts of the theory heavy sections needed some digging to understand and reproduce the implementation.

This project have managed to extend the description of FGR by Zhou et al. 2016 and verify the derivation hinted at in the original article. Mainly the necessary introduction to Lie Groups, Lie Algebras and the exponential map was an important addition. Introducing the Lie Algebra $\mathfrak{se}(3)$ simplified the optimisation updates and did clarify the sentence,
"T is updated by applying ξ to T^k using equation 7, then mapped back into the $SE(3)$ group." [Zhou et al. 2016].
The clarification of this line resulted in section 2.1 and was necessary for solving the optimisation.

The algorithm was implemented as it was derived and described through the project, however, performance is not as good as Zhou et al. 2016 described. This lack in performance is primarily due to a lack in programming experience and not the method. The results presented in the testing section show poor results for real-world datasets, even though the original implementation of the method performed well [Zhou et al. 2018].

For the small datasets tested, the implementation completed as part of this project performed quite well, even with addition of large quantities of outliers. The precision was retained through outlier addition, however Gaussian noise additions even on a scale not visible on the renderings gave rise to incorrect results. For large datasets the project implementation written for the project is not performing well. Most surfaces in the skull datasets were correctly aligned but a few of them were not.

The Fast Global Registration algorithm developed by Zhou et al. 2016 is an efficient, precise algorithm. However, it is not trivial to implement, and parts of the theoretical background were brushed over in the original article. It was possible to derive and describe the theoretical background and a simple implementation were completed. The implementation perform well on small datasets where parameters can be tweaked easily. Integrating this algorithm as part of a pipeline for 3D-scanning could be a great step toward an automated system for scanning general objects.

APPENDIX A

Project evaluation

This project did not follow the plan completely. A bunch of problems arose with the testing, parameters needed a lot of tuning and some of the datasets required a very large amount of time to test due to scan sizes. However the original implementations were done on time as well as most of the preliminary chapters of the project. The risk assessment did end up being correct since the testing parts of the project did take more time than anticipated. The project was planned with a lot of additional time in the end for revisions so the conclusion of the project were not in jeopardy.

A bug found in the last week did change a large amount of the testing and result section, along with the conclusions of the project. This caused the need for re-running all tests, which takes a lot of time.

Project plan

The project was completed in the time between August 27th 2018 and January 27th 2019. Table A.1 shows an outline of when the different parts were completed.

Activity	*Planned completion*	*Time of completion*
Hand-in Project Plan	26th of September 2018	17th of September 2018
Theory examination complete	1st of October 2018	1st of October 2018
Testing framework complete	1st of October 2018	20th of October 2018
Poster session	??	–
All Implementations ready	1st of December 2018	18th of January 2019
All tests concluded	17th of December 2018	25th of January 2019
Sections ready for the report	1st of January 2019	23rd of January 2019
Report ready for external proof reading	13th of January 2019	24th of January 2019
Report ready for print	23rd of January 2019	27th of January 2019
Hand-in Final report	27th of January 2019	27th of January 2019
Presentation ready	8th of February	3rd of February
Defence of thesis	10th of February	6th of February

Table A.1: Project plan including expected and actual dates.

A.1 Original plan

This is the original project plan as handed in and approved September 17, 2018.

Motivation

3D scanning of general objects include some of manual labour at the moment. No scans can be done in one sweep since the object will always be held, or rest in a way which obscures part of the object. This project will aim to reduce the manual part by automating the software based alignment of multiple scans of the same object.

Goals

The goal of the project is to implement a global alignment algorithm which can remove the manual process of registration of multiple partial scans of the same object. The ultimate goal of this is to get the algorithm implemented as part of the DTU 3D scanning software. The proposed algorithm is developed by Zhou et al. 2016 and boasts global efficient 3D registration without the need of a local alignment like ICP. A part of the project will be to test the implementation thoroughly to find eventual limitations.

If this goal is met within a reasonable time some optimisations of the 3D scanning process will be explored. Currently a turntable is programmed to scan from some set angles, lets say every 18 degrees for 20 scan angles. However some objects might have concavities which are poorly caught by this method. So dynamically setting the scan direction could provide a better scan, possibly with fewer angles and therefore a lower time investment.

Technical goals

Performance:
The implementation should be able to compute within a reasonable time. Optimal would be real-time for actual data, however within a couple of minutes (2-3 minutes) could be classified as acceptable.

Precision:
The implementation should produce a result of similar quality to manual alignment combined with a couple of iterations of ICP.

Project plan and risk analysis

This section describes the current deadlines and the risk assessment of the individual parts of the project. The risk assessment is based on a 1 to 5 scale, where 1 is almost risk-free and 5 is very uncertain. As seen in Table A.3 no part of the project is deemed high risk since this project does not depend on external data or other factors which is outside the control of the author.

Activity	*Time of completion*
Hand-in Project Plan	26th of September 2018
Theory examination complete	1st of October 2018
Testing framework complete	1st of October 2018
Poster session	??
All Implementations ready	1st of December 2018
All tests concluded	17th of December 2018
Sections ready for the report	1st of January 2019
Report ready for external proof reading	13th of January 2019
Report ready for print	23rd of January 2019
Hand-in Final report	27th of January 2019
Presentation ready	*8th of February
Defence of thesis	*10th of February

Table A.2: Project plan including expected dates. *Not yet determined.

Activity	*Risk*
Hand-in Project Plan	1
Poster session	2
Implementations	3
Testing	3
Report	1
Presentation	1

Table A.3: Risk estimates of the individual parts of the report.

APPENDIX B CPU Information

The CPU information is obtained by running the `lscpu` command. The output from the command is the following.

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                24
On-line CPU(s) list:   0-23
Thread(s) per core:    1
Core(s) per socket:    12
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 79
Model name:            Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
Stepping:              1
CPU MHz:               2499.975
BogoMIPS:              4405.86
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              30720K
NUMA node0 CPU(s):     0-11
NUMA node1 CPU(s):     12-23
```

Bibliography

Written sources

Aanæs, Henrik (2018). "Lecture Notes on Computer Vision". University Course. Technical University of Denmark, course url: http://kurser.dtu.dk/course/2017-2018/02504.

Black, MJ and A Rangarajan (1996). "On the unification of line processes, outlier rejection, and robust statistics with applications in early vision". eng. In: *International Journal of Computer Vision* 19.1, pp. 57–91. ISSN: 15731405, 09205691. DOI: 10.1007/BF00131148.

Christensen, Ole (2012). *Differentialligninger og uendelige rækker*. dan. DTU Matematik, viii, 331 s., ill. ISBN: 9788788764833.

De Berg, Mark et al. (2008). *Computational geometry: Algorithms and applications*. eng. Springer Berlin Heidelberg, pp. 1–386. ISBN: 9783540779735. DOI: 10.1007/978-3-540-77974-2.

Eade, Ethan (2017). "Lie Groups for 2D and 3D Transformations". In: URL: http://www.ethaneade.com/lie.pdf.

Eggert, D. W. et al. (1997). "Estimating 3-D rigid body transformations: a comparison of four major algorithms". eng. In: DOI: 10.1.1.173.2196.

Gallier, Jean (2001). *Geometric methods and applications*. eng. Vol. 38. Springer, 565 s.

Hartley, Richard (2016). "Some notes on Lie Groups". In: *Mathematics of Shapes and Applications*. URL: www2.ims.nus.edu.sg/Programs/016shape/files/richard.pdf.

Khoury, Marc et al. (2017). "Learning compact geometric features". In: *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 153–61.

Nocedal, J. and S. J. Wright (1999). *Numerical optimization*. eng. Springer, 636 s. ISBN: 0387987932.

Olsen, T. and M. Lorenz (2018). "Large-scale computations on Modern GPUs". University Project. https://github.com/MathiasLorenz/Large_scale_project/blob/master/Large_Scale_Project_Report.pdf.

Park, Soon-Yong et al. (2010). "Real-time 3D registration using GPU". eng. In: *Machine Vision and Applications* 22.5, pp. 837–850. ISSN: 14321769, 09328092. DOI: 10.1007/s00138-010-0282-z.

Qiu, Deyuan et al. (2009). "GPU-Accelerated nearest neighbor search for 3d registration". eng. In: DOI: 10.1.1.323.7141.

Rusinkiewicz, S and M Levoy (2001). "Efficient variants of the ICP algorithm". eng. In: *Third International Conference on 3-d Digital Imaging and Modeling, Proceedings*, pp. 145–152. ISSN: 15506185. DOI: 10.1109/IM.2001.924423.

Rusu, Radu Bogdan, Nico Blodow, and Michael Beetz (2009). "Fast Point Feature Histograms (FPFH) for 3D registration". eng. In: *2009 Ieee International Conference on Robotics and Automation*, pp. 3212–3217. ISSN: 10504729. DOI: 10.1109/ROBOT.2009.5152473.

Rusu, Radu Bogdan, Nico Blodow, Zoltan Csaba Marton, et al. (2008). "Aligning point cloud views using persistent feature histograms". eng. In: *2008 Ieee/rsj International Conference on Intelligent Robots and Systems, Iros*, pp. 4650967, 3384–3391. DOI: 10.1109/IROS.2008.4650967.

Tarel, Jean-Philippe and Nozha Boujemaa (1995). "A Fuzzy 3D Registration Method". eng. In: DOI: 10.1.1.300.6992.

Yang, Jiaolong et al. (2013). “Go-ICP: Solving 3D Registration Efficiently and Globally Optimally”. eng. In: *Computer Vision (iccv), International Conference on*, pp. 1457–1464. ISSN: 15505499. DOI: 10.1109/ICCV.2013.184.

Zhou, Qian-Yi et al. (2016). “Fast Global Registration”. eng. In: *Lecture Notes in Computer Science* 9906. Ed. by Bastian Leibe et al., pp. 766–782. ISSN: 16113349, 03029743. DOI: 10.1007/978-3-319-46475-6_47.

Libraries

Guennebaud, Gael, Benoît Jacob, et al. (2010). *Eigen v3*. http://eigen.tuxfamily.org.

Zhou, Qian-Yi et al. (2018). *Open3D: A Modern Library for 3D Data Processing*. Url: open3d.org.

Datasets

CadNav (2019a). *Armadillo Monster 3D Model*. http://www.cadnav.com/3d-models/model-37161.html.

— (2019b). *Giraffe Laying Down 3D Model*. http://www.cadnav.com/3d-models/model-45916.html.

— (2019c). *X-Bike 3D Model*. http://www.cadnav.com/3d-models/model-47044.html.

Olsen, Morten Tange et al. (2016). *The forgotten type specimen of the grey seal [Halichoerus grypus (Fabricius, 1791)] from the island of Amager, Denmark*. DOI: 10.1111/zoj.12426. URL: http://dx.doi.org/10.1111/zoj.12426.

Stanford Computer Graphics Laboratory (1994). *The Stanford Bunny*. Url: http://graphics.stanford.edu/data/3Dscanrep/.

Wilm, Jakob (n.d.). *3D scan of a fox skull*.