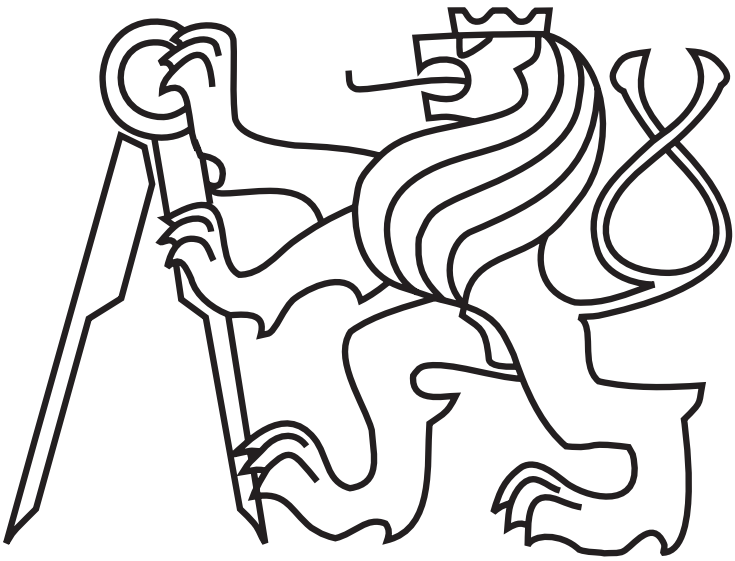
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

DIPLOMA THESIS



Bc. Roman Bedroš

Registration of 3D Point Clouds

Department of Cybernetics

Thesis supervisor: **RNDr. Miroslav Kulich, Ph.D.**

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne..... .....

Podpis autora práce

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student: Bc. Roman B e d r o š

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Diploma Thesis: Registration of 3D Point Clouds

Guidelines:

1. Get acquainted with methods for point clouds registration especially in 3D.
2. Choose, together with the supervisor, one point clouds registration method and implement it.
3. Create an interface of the implemented algorithm for PCL (Point Cloud Library) and integrate the algorithm into this library.
4. Verify experimentally algorithm's behavior with standard robotic datasets, or with data you will collect with robots and sensors at the Department of Cybernetics.
5. Describe the experimental results with respect to speed of the algorithm, its robustness and possible applicability.

Bibliography/Sources:

[1] Makadia, A.; Patterson, A.; Daniilidis, K.: Fully Automatic Registration of 3D Point Clouds. Computer Vision and Pattern Recognition, 2006. IEEE Computer Society Conference on , vol.1, pp.1297,1304, 17-22 June 2006.
[2] Olson, E.B.: Real-Time Correlative Scan Matching. Robotics and Automation, 2009. ICRA '09. IEEE International Conference on , pp.4387,4393, 12-17 May 2009.
[3] Point Cloud Library, http://pointclouds.org, accessed 25.6.2013
[4] Rusinkiewicz, S. and Levoy, M.: Efficient Variants of the ICP Algorithm. 3DIM. 145-152, IEEE Computer Society, 2001.

Diploma Thesis Supervisor: RNDr. Miroslav Kulich, Ph.D.

Valid until: the end of the winter semester of academic year 2014/2015

L.S.

prof. Ing. Vladimír Mařík, DrSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, June 27, 2013

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Roman B e d r o š

Studijní program: Kybernetika a robotika (magisterský)

Obor: Robotika

Název tématu: Metody registrace bodových shluků ve 3D

Pokyny pro vypracování:

1. Proveďte rešerši metod registrace bodových shluků se zaměřením na jejich použití ve 3D.
2. Po dohodě s vedoucím implementujte vybranou metodu registrace 3D bodů.
3. Implementovaný algoritmus opatřete příslušným rozhraním tak, aby mohl být integrován do knihovny PCL (Point Cloud Library).
4. Vlastnosti algoritmu experimentálně ověřte na standardních robotických datových sadách, popřípadě na datech, která naměříte s využitím senzorů a robotů katedry kybernetiky.
5. Experimentální výsledky popište se zaměřením na rychlost, robustnost a možné využití zkoumaného algoritmu.

Seznam odborné literatury:

[1] Makadia, A.; Patterson, A.; Daniilidis, K.: Fully Automatic Registration of 3D Point Clouds. Computer Vision and Pattern Recognition, 2006. IEEE Computer Society Conference on , vol.1, pp.1297,1304, 17-22 June 2006.
[2] Olson, E.B.: Real-Time Correlative Scan Matching. Robotics and Automation, 2009. ICRA '09. IEEE International Conference on , pp.4387,4393, 12-17 May 2009.
[3] Point Cloud Library, http://pointclouds.org, accessed 25.6.2013
[4] Rusinkiewicz, S. and Levoy, M.: Efficient Variants of the ICP Algorithm. 3DIM. 145-152, IEEE Computer Society, 2001.

Vedoucí diplomové práce: RNDr. Miroslav Kulich, Ph.D.

Platnost zadání: do konce zimního semestru 2014/2015

L.S.

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 27. 6. 2013

Acknowledgements

Foremost, I would like to thank to my supervisor, RNDr. Miroslav Kulich, Ph.D. Without his guidance and persistent help this thesis would not have been possible. Also I would like to thank my family for supporting me throughout all my studies.

Abstrakt

Cílem této práce bylo vybrat a implementovat algoritmus pro registraci bodových shluků ve 3D za použití knihovny Point Cloud Library (PCL). Vybraný korelační algoritmus byl detailně popsán, včetně teoretického pozadí. Navíc byly navrženy tři modifikace tohoto algoritmu pro zlepšení jeho vlastností. Tento algoritmus byl implementován spolu s dalšími pomocnými aplikacemi na předzpracování nebo generování vstupních dat. Vlastnosti tohoto algoritmu byly experimentálně testovány na datech z robotického datasetu [1]. Výsledky těchto experimentů ukazují, že je algoritmus schopen registrovat po sobě jdoucí scany zvoleného datasetu za přibližně tři sekundy s chybou ve translaci v řádu centimetrů.

Abstract

A goal of this work was to choose and implement an algorithm for registration of 3D point clouds using the Point Cloud Library (PCL). The selected cross-correlation algorithm was described in detail, including a theory behind it. In addition, three modifications of the algorithm for improving its performance were suggested. The algorithm was implemented along with additional supplementary applications for preprocessing or generating input data. A performance of the algorithm was examined by experiments on robotic dataset [1]. According to these experiments, the algorithm was able to register consecutive scans in approximately three seconds with translational error in order of centimeters for selected dataset.

Contents

List of Figures

Chapter 1

Introduction

Nowadays, a presence of 3D rangefinders (e.g. Microsoft Kinect, ASUS Xtion PRO, Minolta Range 5) in robotic applications is very common, especially on mobile platforms and mounted manipulators. These sensors became very popular especially because of a natural character of acquired data, which is in a certain manner similar to human's perception of a World. Also price and accessibility of rangefinders makes them affordable even for hobbyists. Registration of point clouds, which is a main theme of this work, is one of the most common tasks in processing data acquired by these sensors.

This task can be solved using many algorithms, for example the famous Iterative closest point (ICP) or also very popular Random sample consensus (RANSAC) among others. Each of these algorithms has advantages and drawbacks. The ICP is universal and was proven to converge to local optima, but it needs an estimation of the translation. Similarly the RANSAC is robust, but it requires a feature detection. In this work, the cross-correlation algorithm will be described, implemented and tested, as an alternative to above mentioned ICP and RANSAC.

The main principle of the cross-correlation algorithm lies in determining the rotational part of the searched transformation, which registers assigned clouds, before the translational part. The rotation is computed by generating histograms of normals of the input clouds and finding the rotation for that their cross-correlation is maximal. The translation corresponds to the maximum of cross-correlation of occupancy grids of the clouds. The main advantage of this algorithm is that it finds a global optimum of the cross-correlation of the input scans, so neither estimation of the transformation nor feature detection is needed.

The work is structured as follows. A definition of the registration task is formalized in chapter 2 and an overview of algorithms often used for finding its solution is given in chapter 3. In chapter 4, the cross-correlation algorithm is described in detail. An implementation of the algorithm itself, along with other created additional tools, is described in chapter 5. In chapter 6, conducted experiments and observed properties of the implemented algorithm are described.

Chapter 2

Problem definition

In this chapter, the task of registration of point clouds is introduced. First, in chapter 2.1, basic definition of point clouds is given, then, in chapter 2.2, the registration task is formally defined.

2.1 Point clouds

In this text, a point cloud $\mathcal{C}$ is defined as a finite sequence of points

$$\mathcal{C} = (C_i)_{i=1}^{n} = (C_1, C_2, C_3, \ldots, C_n),$$

where a position of each point C_i in a 3D space is defined by three coordinates $C_i = \begin{bmatrix} C_1^x \\ C_1^y \\ C_1^z \end{bmatrix}$. Coordinates of points in the cloud can be also written as a matrix

$$C = \begin{bmatrix} C_1^x & C_2^x & C_3^x & \ldots \\ C_1^y & C_2^y & C_3^y & \ldots \\ C_1^z & C_2^z & C_3^z & \ldots \end{bmatrix} \tag{2.1}$$

which is useful in some expressions. Example of a point cloud containing three points is in the Figure 2.1. Figure 2.2 shows an example of a cloud obtained by a real sensor (in this case ASUS Xtion PRO).

Rangefinders usually acquire clouds with frequency 1-30 Hz, which can lead to inaccuracies caused by a movement of the sensor or scanned objects. In this work we will neglect this type of error assuming that the sensor is fast enough when capturing the data.

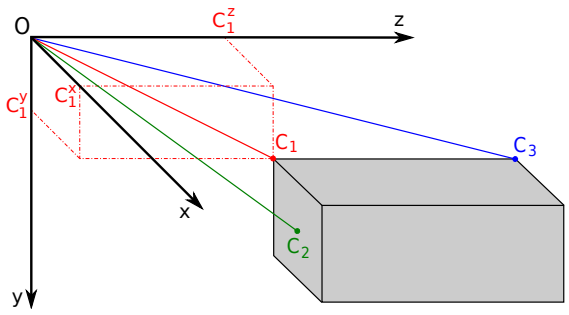


Figure 2.1: A point cloud containing three points lying on a surface of a grey box. A sensor is placed in point O.

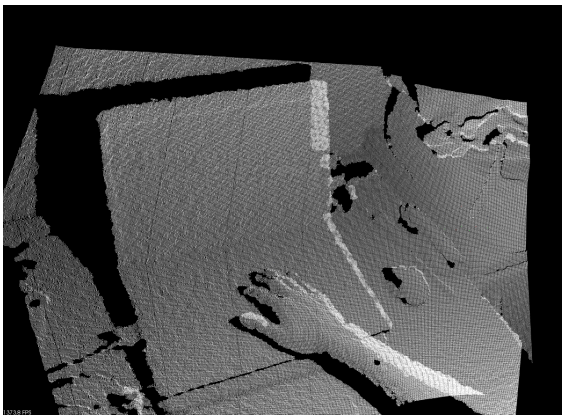


Figure 2.2: An example of a point cloud scanned by a real sensor. This cloud consists of 307200 (i.e. 640×480) points.

2.2 Registration

Registration of point clouds is a task with two input point clouds: $\mathcal{T}$ (target) and $\mathcal{S}$ (source), which typically partially overlap. Simply put, it is a process of finding a way how to move the source cloud such that overlapping parts of both scans were aligned.

Formally defined, a goal of the registration is to find a transformation which maximizes a value of similarity function F_{sim} of the cloud $\mathcal{T}$ and a transformed cloud $\mathcal{S}$. Searched transformation consists of a rotation (which can be described by a rotation matrix $R \in \mathbb{R}^{3\times3}$) and a translation (which can be described by a vector $Q \in \mathbb{R}^{3\times1}$), so the solution of the task can be mathematically expressed as

$$\{R_{sol}, Q_{sol}\} = \arg\max_{R,Q} F_{sim}(T, RS + Q). \tag{2.2}$$

The similarity function F_{sim} can be defined in more ways, in this work, a discrete inner product of occupancy grids of clouds will be used. The occupancy grid of cloud $\mathcal{C}$ is created by dividing a space (in this case a 3D area) into cubes (so called voxels) and determining a value corresponding to each of these cubes. The value of a voxel equals to a number of points of cloud $\mathcal{C}$ lying in the voxel. The voxels can be ordered and indexed one by one, in this case, a value of the i-th voxel of $\mathcal{C}$ will be denoted as c_i, where $i \in \mathbb{N}$. Another possibility is to index voxels in all three dimmensions separately. For this case, a notation $c_{x,y,z}$, where $x, y, z \in \mathbb{N}$ will be used.

The similarity function F_{sim} of clouds T and $W = RS + Q$ is then defined as follows

$$F_{sim}(T, W) = \sum_{i=1}^{m} t_i \cdot w_i, \tag{2.3}$$

where t_i is a value of the i-th voxel of cloud T and w_i is a value of the i-th voxel of cloud $W = RS + Q$.

Chapter 3

Registration algorithms

A lot of methods using different approaches were developed for registration of point clouds. In this section, some of the most used ones are introduced. A cross-correlation algorithm, implemented as a part of this work, is in this chapter described only briefly, more detailed description can be found in chapter 4.

3.1 Iterative closest point

The iterative closest point (ICP) algorithm, introduced in [3], is one of the most popular algorithms for registration of point clouds. This algorithm is very universal, because in addition to point clouds, it is able to register the source cloud to some other representations of the target geometric data, such as line segment sets, triangle sets, curves or surfaces. The only condition is that it must be possible to find a target point that has the smallest Euclidean distance from the selected source point. It is possible to register both, 2D and 3D geometric data with the ICP. In addition to that, a convergence of this algorithm was proven in [3].

The main disadvantage of the ICP algorithm is a fact that it converges only to a local optima. It means that if the searched transformation between the target and source clouds is not small enough, the algorithm can compute a wrong transformation. Therefore for general usage, an estimation of the searched transformation is needed, which is, in case of mobile robots, usually obtained from odometry. The ICP algorithm is also sensitive to the relative size of an overlapping area, because all points that don't have correspondence cause an error in the registration.

As described in [2], the ICP is an iterative algorithm that performs following tasks in each iteration.

1. Cloud $\mathcal{S}'$ is generated by applying a transformation computed in the previous iteration to source cloud $\mathcal{S}$ (in the first iteration, the odometry estimation is used, or an identity if no estimation is known).

2. For each point $S'_i \in \mathcal{S}'$, a corresponding point $T_i \in \mathcal{T}$ is found such that T_i is the closest point to S'_i of all points in $\mathcal{T}$.

3. From sets of corresponding points S'_i and T_i a new transformation that minimizes a penalization function 3.1 is computed.

Iterating is stopped when the new transformation differs from the transformation computed in the previous iteration negligibly (usually if the difference between two consecutive transformations is lower than a predefined threshold). Ilustration of the tasks executed in one iteration of the algorithm is in Figure 3.1.

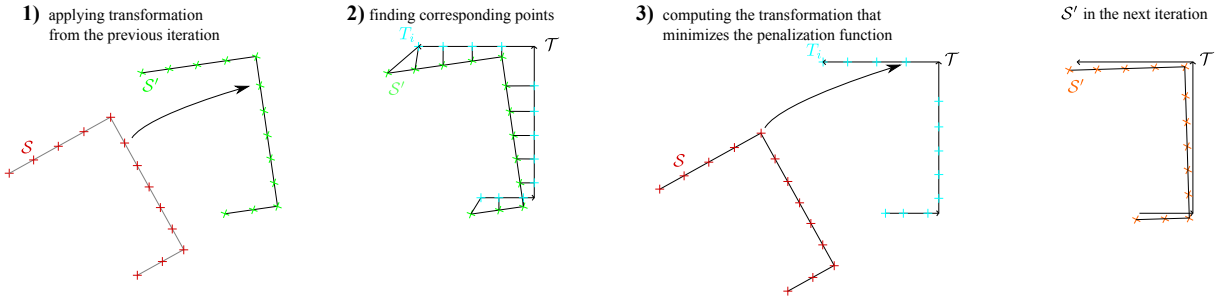


Figure 3.1: One iteration of the ICP algorithm for 2D data. The source cloud contains 13 points and the target consists of three line segments. Excerpted from [2].

The penalization function minimized in step 3 of the ICP algorithm is a sum of squares of the Euclidean distances between points S'_i and corresponding points T_i, which can be mathematically written as

$$E(R, Q) = \sum_i |RS'_i + Q - T_i|^2, \tag{3.1}$$

where $R \in \mathbb{R}^{3\times3}$ is a rotation matrix and $Q \in \mathbb{R}^{3\times1}$ is a translation vector. The transformation which minimizes this penalization function can be computed according to an algorithm described in [3].

A lot of variants and improvements of this algorithm were developed. A basic overview of them including comparison of their performance can be found in [4]. Some more recent papers even focus directly on usage of this algorithm with popular rangefinders. For example [5] formulates the ICP in inverse depth coordinates which better conform to a way in which these sensors acquire data.

3.2 Random sample consensus

The random sample consensus (RANSAC) algorithm, originally introduced in [6], is a non-deterministic algorithm for a robust model fitting. It is able to find parameters of a model of data from given set $\mathcal{Z}$ of samples containing outliers. The RANSAC is an iterative algorithm that performs following steps in each iteration.

1. A subset of samples $\mathcal{Z}'$ is randomly chosen from $\mathcal{Z}$.
2. Parameters of a model which fits to $\mathcal{Z}'$ are computed.
3. Number of inliers (points of $\mathcal{Z}$ that are closer to the model than a predefined threshold) is computed.
4. If the number of inliers is higher than the number of inliers of the best model found so far, actual model becames the best one.

Iterating is finished after predefined number of iterations.

Usage of the RANSAC algorithm for registration was described in [7] among others. For registration tasks, the RANSAC algorithm is usually slightly modified. It is preceded with a keypoint detection (e.g. SIFT, SURF or ORB keypoints are used) and instead of choosing points, couples of keypoints (one source and one target) are selected. One iteration of the algorithm then consists of following steps.

1. At least three (minimum for computing a transformation) couples of keypoints are randomly selected.
2. A transformation that transforms source keypoints to positions of their coupled target keypoints is computed.
3. Number of inliers (transformed source keypoints that are close enough to some target keypoint) is computed.
4. If the number of inliers is higher than the number of inliers of the best transformation found so far, the actual transformation becames the best one.

The main disadvantage of this algorithm is a fact that it requires a robust keypoint detection, which can be problematic in some environments.

3.3 Cross-correlation

In some applications, use of the reliable ICP algorithm isn't possible, usually because of unknown estimation of a transformation or too small overlap of target and source clouds. In these cases, it is required to find a global maximum of the similarity function F_{sim} (defined in 2.2). If a keypoint detection is possible, it is a perfect task for the RANSAC algorithm. But if the detection can't be done, a different approach must be used. One way of solving these tasks is to use an algorithm that computes whole cross-correlation of occupancy grids.

The main problem of this approach is that a space of possible transformations (matrices R, Q in 2.2) is enormous. Namely a rotational part of the transformation can be defined by three independent variables (e.g. Euler angles) and a translational part of the transformation by

another three independent variables. So that even for a resolution 2° in each angle of rotation and 50 discrete steps in each axis of translation, the number of possible transformations is $180^3 \cdot 50^3 = 729 \cdot 10^9$. If a naive approach had been used, an occupancy grid would have been created and a similarity function 2.3 evaluated for each of these transformations.

A lot of techniques were developed for avoiding this computationally demanding step. One of them lies in dividing the transformation into a rotational and a translational part and computing these parts separately. This technique (concretely determining the rotation first and the translation afterwards) is used in the algorithm, which was implemented as a part of this work and which is described narrowly in chapter 4.

Chapter 4

Cross-correlation algorithm

This chapter contains a detailed description of the cross-correlation algorithm briefly introduced in [8], which was implemented as a part of this work. At first, an overview of the algorithm, including a pseudo-code, is given in section 4.1. Following sections 4.2 and 4.3 describe individual steps of the pseudo-code with necessary mathematical background. Finally, section 4.4 contains descriptions of modifications of the original algorithm that were implemented in order to improve its properties or enable usage of the algorithm in specific applications.

4.1 Overview of the algorithm

An input of the cross-correlation algorithm consists of target ($\mathcal{T}$) and source ($\mathcal{S}$) clouds introduced in chapter 2.2.

Outputs of the algorithm are rotation R_{sol} and translation Q_{sol} from equation 2.2. R_{sol} and Q_{sol} maximize the similarity function f_{sim} defined by equation 2.3. A principle of the algorithm can be described by the following scheme.

1. rotation R_{sol} is determined, as described in chapter 4.2
   (a) centres $\mathcal{H} = (H_i)_{i=1}^{h} = \left(\begin{bmatrix}\theta_i \\ \phi_i\end{bmatrix}\right)_{i=1}^{h}$ of histogram bins (points uniformly distributed on a unit sphere) are generated using formulas 4.1 and 4.2
   (b) histograms $(t_i)_{i=1}^{h}$ of the target cloud and $(s_i)_{i=1}^{h}$ of the source cloud are computed from $\mathcal{T}$, $\mathcal{S}$ and $\mathcal{H}$ using algorithm 1
   (c) coefficients $\hat{t}_l^m$ and $\hat{s}_l^m$ of a spherical harmonic (SPH) expansion of the target and source histograms $(t_i)_{i=1}^{h}$ and $(s_i)_{i=1}^{h}$ are computed for all bands l up to selected maximum l_{max}
      i. values of associated Legendre polynomials $P_l^m(\cos\theta_i)$ for $l \leq l_{max}$ are computed using formulas 4.4a - 4.4d

ii. values of spherical harmonic functions $Y_l^m(\theta_i, \phi_i)$ in points H_i are computed from $P_l^m(\cos\theta_i)$ according to equation 4.5
iii. coefficients $\hat{t}_l^m$ and $\hat{s}_l^m$ are computed from $(t_i)_{i=1}^h$, $(s_i)_{i=1}^h$ and $Y_l^m(H_i)$ using formula 4.13

(d) cross-correlation $C(R_i)$ of the target and source histograms is computed from $\hat{t}_l^m$ and $\hat{s}_l^m$
i. samples $(R_i)_{i=1}^r = \left(\begin{bmatrix}\alpha_i\\ \beta_i\\ \gamma_i\end{bmatrix}\right)_{i=1}^r$ of a space of 3D rotations around origin are selected
ii. values of Wigner d-function $d_{m,n}^l(\beta_i)$ are computed for all β_i using equation 4.28
iii. values of Wigner D-function $D_{m,n}^l(R_i)$ are computed for all samples R_i from α_i, γ_i and $d_{m,n}^l(\beta_i)$, using equation 4.24
iv. from $\hat{t}_l^m$, $\hat{s}_l^m$ and $D_{m,n}^l(R_i)$, a value of cross-correlation $C(R_i)$ is computed for all samples R_i according to formula 4.23

(e) rotation $R_{sol} = \arg\max_{R_i} |C(R_i)|$ is found as the rotation for that a magnitude of cross-correlation $C(R_i)$ reaches its maximum

2. translation Q_{sol} is determined, as described in chapter 4.3

(a) rough translation Q' is determined from $\mathcal{T}$, $\mathcal{S}$ and R_{sol}
i. points T_{min} and U_{min} defined by equation 4.29 are found
ii. translation Q' is computed from T_{min} and U_{min} according to equation 4.30

(b) fine translation Q'' is determined from $\mathcal{T}$, $\mathcal{S}$, R_{sol} and Q'
i. occupancy grids $t_{x,y,z}$ of the target cloud T and $v_{x,y,z}$ of cloud $V = R_{sol}S + Q'$ are created from $\mathcal{T}$, $\mathcal{S}$, R_{sol} and Q'
ii. direct DFT is applied to $t_{x,y,z}$ and $v_{x,y,z}$ producing coefficients $\mathcal{F}(t)_{x,y,z}$ and $\mathcal{F}(v)_{x,y,z}$ according to equation 4.32
iii. coefficients of a correlation $\mathcal{F}(t \star s)$ are computed from $\mathcal{F}(t)_{x,y,z}$ and $\mathcal{F}(v)_{x,y,z}$ using formula 4.31
iv. correlation $(t \star s)(a, b, c)$ of occupancy grids $t_{x,y,z}$ and $v_{x,y,z}$ is computed from $\mathcal{F}(t \star s)$ using an inverse 3D DFT defined by 4.33
v. points a_{max}, b_{max}, c_{max} for which $(t \star s)(a, b, c)$ reaches its maximum are found
vi. translation Q'' is computed from a_{max}, b_{max}, c_{max} according to equation 4.34

(c) translation Q_{sol} is computed using equation 4.35

4.2 Rotation determination

Rotation R_{sol} from equation 2.2 is determined in step 1 of the pseudo-code in section 4.1. This part of the algorithm serves for finding a rotational component (matrix R_{sol}) of the

searched transformation regardless of a translation. A principle of this approach lies in finding a *histogram of normals* of target and source clouds, computing their cross-correlation (using an *spherical harmonic expansion* and an $SO(3)$ *Fourier transform*) and finding a rotation for which the cross-correlation reaches its maximum. Details of the mentioned procedures and mathematical concepts will be introduced in following sections.

The spherical harmonic (SPH) transformation and the $SO(3)$ Fourier transform (SOFT) are mathematical concepts based on principles described in [9]. This mathematical theory was summed up and extended (including practical applications) in [10], [11], [12] among others.

4.2.1 Spherical coordinate system

Before explaining the methods for computing the rotation, a spherical coordinate system will be defined here. It is necessary, because more conventions in diferent sources exist, which can cause misunderstandings.

The spherical coordinate system (which is an alternative of the Cartesian one) is a coordinate system which describes a position of a point X in a 3D space with an origin O by 3 coordinates: θ, ϕ, r. For this work, the following convention of coordinates will be used.

$\phi \in [0, 2\pi)$ is an angle between an x-axis and a projection of a line segment OX onto a plane xy. $\theta \in [0, \pi]$ is an angle between a z-axis and a line segment OX. Finally, r is an Euclidean distance between points O and X. Ilustration of a spherical coordinate system defined such way is in Figure 4.1.

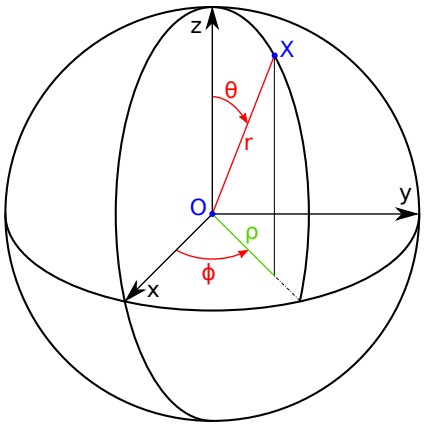


Figure 4.1: Spherical coordinate system.

Following relations between Cartesian coordinate system and the spherical one hold:

$$\begin{aligned} \rho &= \sqrt{x^2 + y^2} & \rho &= r\sin(\theta) \\ r &= \sqrt{x^2 + y^2 + z^2} & x &= \rho\cos(\phi) \\ \phi &= \arctan\left(\frac{y}{x}\right) & y &= \rho\sin(\phi) \\ \theta &= \arcsin\left(\frac{\rho}{r}\right) = \arccos\left(\frac{z}{r}\right) & z &= r\cos(\theta). \end{aligned}$$

4.2.2 Histogram of normals

The histogram of normals is generated in step 1b of the pseudo-code in section 4.1. In this part of the algorithm, we try to eliminate a translational part of searched transformation, so a proper description of clouds, which isn't affected by a translation, must be used. The histogram of normals is one of such descriptions. It describes a cloud by a real function $f(\theta, \phi)$ defined on discrete points of a unit sphere, where θ and ϕ are spherical coordinates of a discrete point (r is always equal to one). The discrete points represent centres of bins of the histogram and their positions must be chosen before computing the histogram itself. The process of selection of these points will be described later in this chapter.

Besides positions of bins, it is also needed to compute surface normals in each point of the described cloud. For this task a method introduced in [13] can be used.

Having both, centres $\mathcal{H} = (H_i)_{i=1}^{h}$ of bins, and normals $(\vec{n}_j)_{j=1}^{l}$ of the described cloud, the histogram can be computed according to algorithm 1.

Algorithm 1: Generating a histogram of normals.

input : centres $\mathcal{H} = (H_i)_{i=1}^{h}$ of histogram bins, normals $(\vec{n}_j)_{j=1}^{l}$ of the described cloud
output: histogram $(h_i)_{i=1}^{h}$ of the described cloud

```
for i = 1 to h do
    h_i ← 0;
end
for j = 1 to l do
    translate the normal n_j such that its starting point lies in an origin of the coordinate
    system;
    find endpoint N_j of the translated normal n_j;
    find point H_x ∈ H with a minimal Euclidean distance from point N_j;
    h_x ← h_x + 1;
end
```

A histogram created this way can be visualized in many ways, one of them displays centres of bins on a unit sphere and values of bins are represented by a colour of points (brighter colour means higher value of a bin). Another possibility is to represent values of individual bins by

distance of center points from origin of a coordinate system. Both these methods are illustrated in Figure 4.4. A cloud from which the histogram was generated is in Figure 4.3.

Some further computations (concretely discrete integration using a Monte Carlo estimator described by equation 4.12) will require the bins' centres to be placed uniformly, independently of a parametrization of the sphere used (in our case angles θ, ϕ). An intuitive approach consisting in the use of formulas

$$\theta = u\pi$$
$$\phi = v2\pi,$$

where $u \in [0, 1]$ and $v \in [0, 1)$ are uniformly distributed, leads to an incorrect solution, because generated points are accumulated near poles of the sphere (which is apparent in Figure 4.5 in comparison to Figure 4.6). For correct generating of points uniformly distributed over the sphere, one of methods listed in [14] can be used. The simplest one uses formulas

$$\theta = \arccos(2u - 1) \tag{4.1}$$
$$\phi = v2\pi, \tag{4.2}$$

where u and v are uniformly distributed over the interval $(0, 1)$. Another method, described in [8], uses a projection of a polyhedra (concretely an icosahedron with further divided faces) onto a sphere.

4.2.3 Associated Legendre polynomials

Associated Legendre polynomials are computed in step 1(c)i of the pseudo-code in section 4.1. They are introduced here, because they will be used in chapter 4.2.4 for defining spherical harmonic functions.

Associated Legendre polynomials are real functions of real value, defined, according to [15], as solutions to the associated Legendre differential equation. It is possible to express them explicitly by a formula

$$P_l^m(x) = (-1)^m \sqrt{(1 - x^2)^m} \left(\frac{\mathrm{d}m}{\mathrm{d}x^m} P_l(x) \right), \tag{4.3}$$

where $l \in \mathbb{N}_0$ is a band (sometimes called a degree), $m \in (0, 1, 2, \ldots, l)$ denotes an order (it doesn't indicate a power of P_l) and $P_l(x)$ is an unassociated Legendre polynomial defined in [16] as

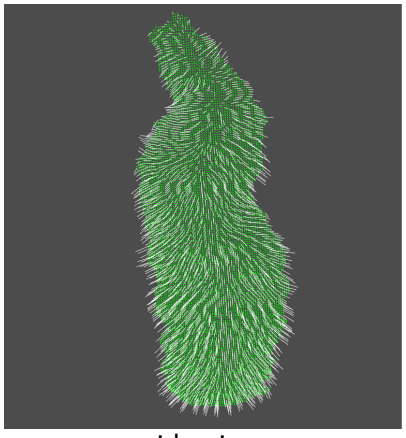
$$P_l(x) = \frac{1}{2^l l!} \frac{\mathrm{d}^l}{\mathrm{d}x^l} (x^2 - 1)^l.$$

For negative order, associated Legendre polynomials are defined by a formula 4.4d.

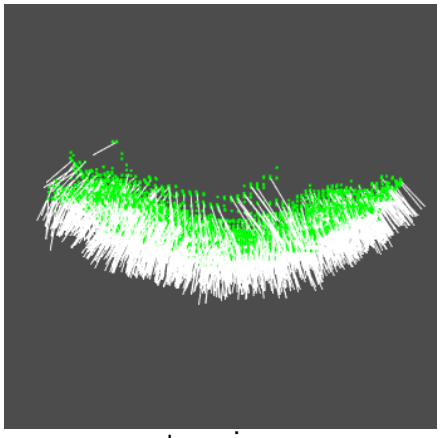
photo of the bottle

scanned cloud

Figure 4.2: A plastic bottle that was scanned with ASUS Xtion PRO.

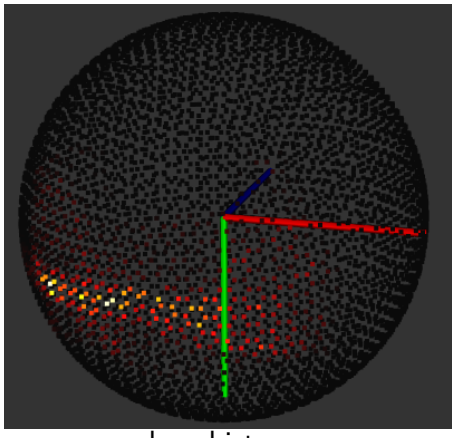


side view

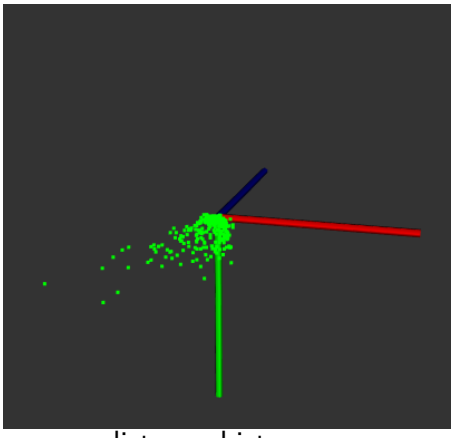


top view

Figure 4.3: Scan of a plastic bottle from figure 4.2. All points and each 5-th normal are displayed.

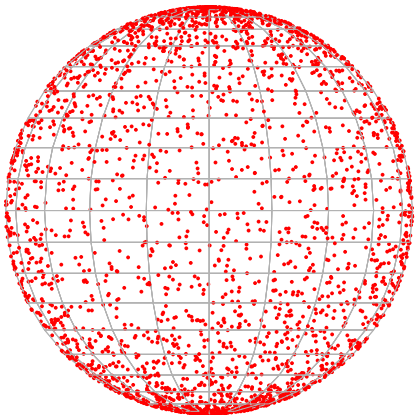


colour histogram

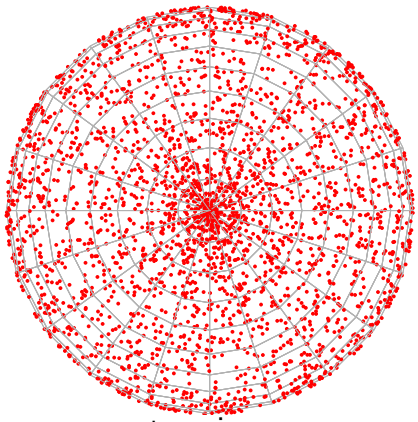


distance histogram

Figure 4.4: Histogram of plastic bottle from Figure 4.3.

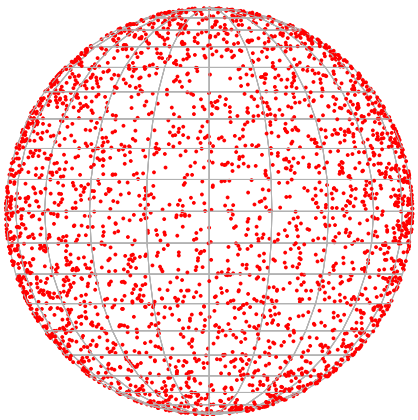


side view

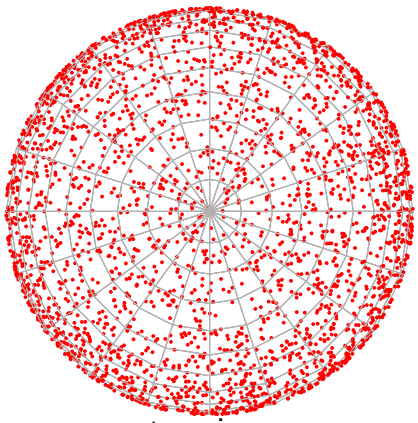


top view

Figure 4.5: Intuitively sampled sphere.



side view



top view

Figure 4.6: Uniformly sampled sphere.

Although formula 4.3 is explicit, it is not very practical for numerical computation, because it contains derivations. Fortunately, associated Legendre polynomials can be also computed using recurrence relations

$$P_l^l(x) = (-1)^l(2l-1)!!\sqrt{(1-x^2)^l} \tag{4.4a}$$

$$P_{l+1}^l(x) = x(2l+1)P_l^l(x) \tag{4.4b}$$

$$(l-m)P_l^m(x) = x(2l-1)P_{l-1}^m(x) - (l+m-1)P_{l-2}^m(x) \tag{4.4c}$$

$$P_l^{-m}(x) = (-1)^m\frac{(l-m)!}{(l+m)!}P_l^m(x), \tag{4.4d}$$

where $m \in (0, 1, 2, \ldots, l)$ and symbol !! denotes a double factorial defined as

$$(n)!! = \begin{cases} n \cdot (n-2) \cdot \ldots \cdot 5 \cdot 3 \cdot 1 & \text{for } n > 0 \text{ odd} \\ n \cdot (n-2) \cdot \ldots \cdot 6 \cdot 4 \cdot 2 & \text{for } n > 0 \text{ even} \\ 1 & \text{for } n = -1, 0, \end{cases}$$

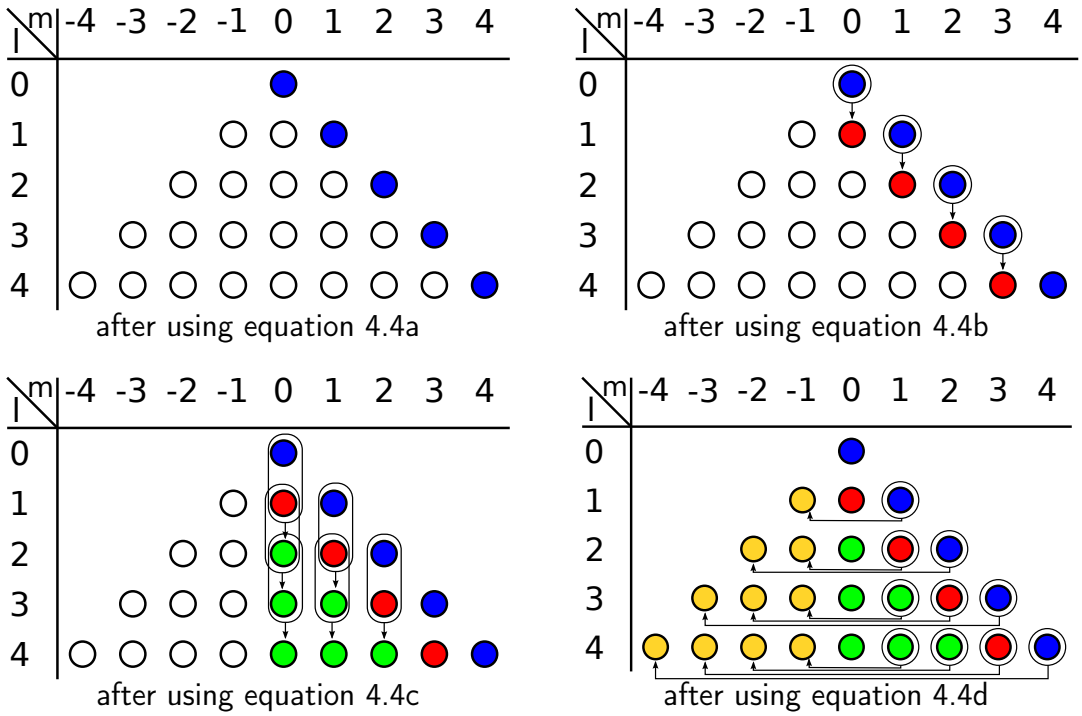


Figure 4.7: Recursive computation of associated Legendre polynomials. White circles indicate currently unknown values, coloured ones represent values already computed. Input values are in bubbles, outputs are pointed by arrows.

so equation 4.3 doesn't need to be used at all. The recursive method of calculation is advantageous especially if the whole set

$$\mathcal{P} = \{P_l^m(x) \,|\, l < l_{max}; -l \leq m \leq l\}$$

for a given band l_{max} and argument x is computed at once, which is the case of the algorithm described in this chapter. The recursive computational process is illustrated in Figure 4.7.

4.2.4 Spherical harmonics (SPH)

Similarly to a spherical coordinate system, spherical harmonic functions (or shortly spherical harmonics), computed in step 1(c)ii of the algorithm, are defined inconsistently in different sources (e.g. [11], [12], [17]). The most common way is to define them as the angular portion of the solution to Laplace's equation in spherical coordinates. However, this definition isn't very practical for computing and what is more, it contains an urcentainty consisting in a normalization coefficient.

Therefore, in this work an explicit definition of spherical harmonics, similar to the one used in [17], will be used. According to it, the spherical harmonics are functions $f : \mathbb{S}^2 \to \mathbb{C}$, that

can be written as

$$\boxed{Y_l^m(\theta,\phi) = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}}P_l^m(\cos\theta)e^{im\phi}}, \tag{4.5}$$

where $\mathbb{S}^2$ denotes a unit sphere, $\theta \in [0,\pi]$ and $\phi \in [0,2\pi)$ are spherical coordinates defining a point on the unit sphere, $l \in \mathbb{N}_0$ denotes a band (or degree), $m \in (-l,\ldots,0,\ldots,l)$ denotes an order of the spherical harmonic (as for associated Legendre polynomials, it doesn't indicate a power of Y_l), P_l^m is an associated Legendre polynomial and i is an imaginary unit. Figure 4.8 displays real parts of spherical harmoics up to band 2.

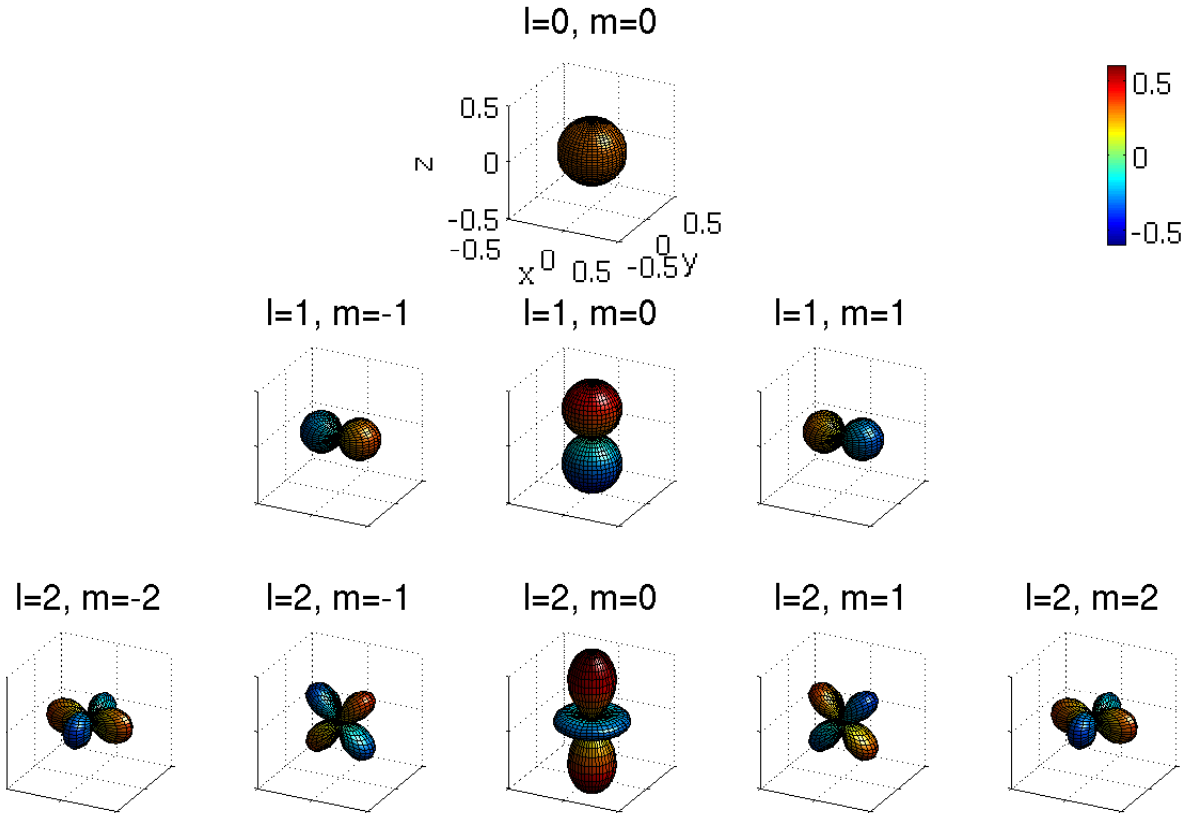


Figure 4.8: Spherical harmonics of bands 0-2. Axis (including scales) are the same for all functions.

Sometimes it is advantageous to use only one number for indexing spherical harmonics instead of two (l and m). In these cases, notation $Y_i(\theta,\phi)$ denoting the same function as $Y_l^m(\theta,\phi)$ can be used, where conversion formula is

$$i = (l+1)l + m.$$

This conversion can be interpretted as ordering spherical harmonics by l from lowest to highest and then by m (from $-l$ to l) in each band. Index i can then be generated by indexing such

ordered functions from 0. Both ways of indexing spherical harmonics are illustrated in Figure 4.9.

l \ m	-3	-2	-1	0	1	2	3
0				0			
1			1	2	3		
2		4	5	6	7	8	
3	9	10	11	12	13	14	15

Figure 4.9: Two ways of indexing spherical harmonics. The first one uses indices l (green) and m (red), the second one uses only index i (yellow).

4.2.5 Spherical harmonic expansion and reconstruction

As a first approach, it is possible to imagine SPH expansion and reconstruction as operations in some way similar to very popular direct and inverse Discrete Fourier transform (DFT). The direct DFT projects a function onto a basis consisting of selected goniometric functions, generating a list of complex coefficients. Similarly, the SPH expansion (computed in step 1c of the algorithm) projects a function onto a basis consisting of spherical harmonics, generating a list of complex coefficients (referred as SPH coefficients). The SPH reconstruction is an inverse operation for SPH expansion, much like the inverse DFT is for direct DFT.

For a formal definition, let's first denote a space of square-integrable complex spherical functions as $L^2(\mathbb{S}^2)$. Then the SPH expansion of a function $f \in L^2(\mathbb{S}^2)$ generates a list of complex coefficients

$$\hat{f} = (\hat{f}_0^0, \hat{f}_1^{-1}, \hat{f}_1^0, \hat{f}_1^1, \dots) = (\hat{f}_0, \hat{f}_1, \hat{f}_2, \hat{f}_3, \dots)$$

defined as

$$\hat{f}_l^m = < f, Y_l^m > \tag{4.6}$$

or using one-index notation

$$\hat{f}_i = < f, Y_i >, \tag{4.7}$$

where $\hat{f}_l^m = \hat{f}_i \in \mathbb{C}$, $l \in \mathbb{N}_0$ denotes a band, $m \in (-l, \dots, 0, \dots, l)$ denotes an order and $<>$ denotes an inner product, which is for functions $L^2(\mathbb{S}^2)$ defined as

$$< f, g >= \int_S f(\omega)\overline{g}(\omega)\mathrm{d}\omega = \int_0^{2\pi}\int_0^{\pi} f(\theta,\phi)\overline{g}(\theta,\phi)\sin(\theta)\mathrm{d}\theta\mathrm{d}\phi, \tag{4.8}$$

where $\int_S$ denotes an integral over a sphere and $\overline{g}$ denotes a complex conjugate of g. Combining equations 4.6 and 4.8, the following formula for computing coefficients of the SPH expansion can be derived

$$\hat{f}_l^m = \int_0^{2\pi} \int_0^{\pi} f(\theta, \phi) \overline{Y_l^m}(\theta, \phi) \sin(\theta) \mathrm{d}\theta \mathrm{d}\phi. \tag{4.9}$$

Function $f : L^2(\mathbb{S}^2)$ can be, according to [11], reconstructed from coefficients $\hat{f}_i$ using formula

$$f(\theta, \phi) = \sum_{i=0}^{\infty} \hat{f}_i \, Y_i(\theta, \phi) \tag{4.10}$$

or from coefficients $\hat{f}_l^m$ using formula

$$f(\theta, \phi) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \hat{f}_l^m \, Y_l^m(\theta, \phi), \tag{4.11}$$

which represent a linear combination of all base functions (spherical harmonics) multiplied by their associated coefficients.

So far, using formulas 4.9 and 4.10, we are able to expand and reconstruct functions from $L^2(\mathbb{S}^2)$. Formula 4.10 can also be used for reconstructing sampled (or "discrete") spherical functions (i.e. defined only on specified points of a unit sphere), but neither formula 4.9 nor 4.6 can deal with this type of functions directly. For expansion, integral over a sphere in these formulas must be discretized and values of the sampled function estimated in areas between sampling points. This can be achieved by so called Monte Carlo estimator described in [11] and [18]. According to it, an integral over multidimensional volume V can be estimated as

$$\int h \mathrm{d}V \approx \frac{V}{N} \sum_{s=1}^{N} h(X_s), \tag{4.12}$$

where N is a number of samples and $X_1, X_2, \ldots, X_N$ are samples uniformly distributed in V. Using this formula with V substitued by a surface of a unit sphere, an inner product (defined by 4.8) can be estimated as follows

$$< f, g >= \int_S f(\omega) \overline{g}(\omega) \mathrm{d}\omega \approx \frac{4\pi}{N} \sum_{s=1}^{N} f(X_s) \overline{g}(X_s),$$

so a formula for estimating coefficients of an SPH expansion of uniformly sampled spherical function f is

$$\boxed{\hat{f}_l^m \approx \frac{4\pi}{N} \sum_{s=1}^{N} f(X_s) \overline{Y_l^m}(X_s)}. \tag{4.13}$$

When working with real data, formula 4.11 can't be used as is, because it contains an infinite sum which can't be computed in a finite time. Instead, an approximation can be computed,

using the following formula

$$\boxed{f(\theta,\phi) = \sum_{l=0}^{l_{max}} \sum_{m=-l}^{l} \hat{f}_l^m \, Y_l^m(\theta,\phi),} \tag{4.14}$$

where $l_{max} \in \mathbb{N}_0$ is a maximal band. A value of l_{max} is selected as a trade-off between a precision of the approximation and its speed. But this approximation can be exact for some functions. If the expanded function does not contain "high frequency content" (i.e. it has all SPH coefficients with $l > l_{max}$ equal to zero), nothing is lost during the approximated reconstruction by equation 4.14 and the original function is reconstructed exactly. But if a function with a "high frequency content" is expanded, reconstruction according to equation 4.14 generates only an approximation of the original expanded function. However this approximation can be often sufficient. An effect of the approximation is illustrated in Figure 4.10.

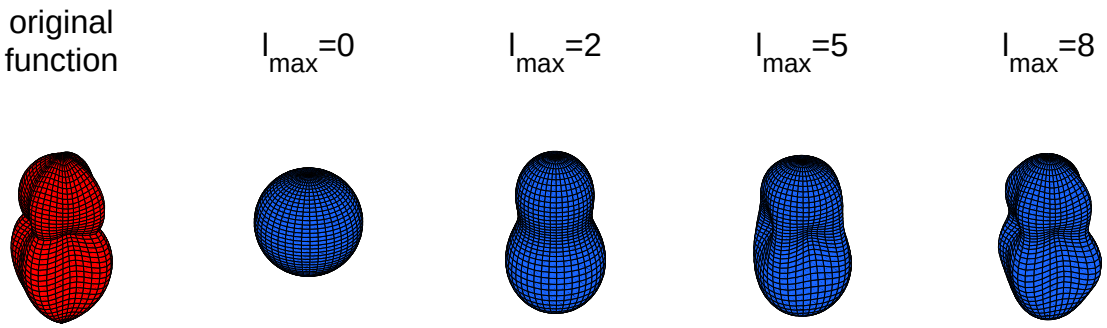


Figure 4.10: Effect of filtering high frequency content out. Red function is expanded and then reconstructed using only coefficients from 0 to l_{max}.

4.2.6 Notations of rotation

A rotation in 3D space can be described in many ways, and of course, some of them are more advantageous in some cases than the others. In this work, three kinds of notation are used, all of them are described in this chapter, along with some relations between them.

Rotation matrix

The first notation, the rotation matrix is a 3×3 real matrix R with $\det(R) = 1$. Further, $R^{-1} = R^T$, where R^{-1} denotes an inverse of R and R^T is matrix R transposed.

This notation is very handy when rotating clouds, because if an input cloud C is expressed by the equation 2.1, nothing more than a simple matrix multiplication $D = RC$ is needed for generating a rotated cloud D. The next advantage of the rotation matrix is an easy chaining

of consecutive rotations. If a cloud C is rotated by a rotation R_1 and then by another rotation R_2, it can be written as

$$R_2 R_1 C = R_{2,1} C. \tag{4.15}$$

So a single rotation consisting of 2 consecutive rotations can be enumerated by a matrix multiplication.

On the other hand, the rotation matrix has also drawbacks. At first, it isn't very illustrative (i.e. it is difficult to interpret nine numbers as an imaginable rotation without a visualization tool). The second disadvantage is that the rotation matrix is very redundant, which can be a problem when storing larger number of them. The redundance is also a problem for random generating, because all nine numbers depend on each other.

Euler angles

The next notation, which uses so called Euler angles, expresses a 3D rotation as three independent numbers α, β, γ. These numbers represent three consecutive rotations around selected axis of a coordinate system. Axes can be selected in more ways, but in this work, so called z-y-z convention will be used consistently. According to this convention, an input cloud C and axes x, y, z are firstly rotated around axis z by angle α, producing cloud C' and new axes x', y', z' (so axes $z \equiv z'$). Then cloud C' and axes x', y', z' are rotated around axis y' by angle β, producing a cloud C'' and axes x'', y'', z'' (so axes $y' \equiv y''$). Finally cloud C'' is rotated around axis z'' by angle γ, producing an output cloud.

A formula for converting Euler angles α, β, γ to rotation matrix $R_{zyz}(\alpha, \beta, \gamma)$, can be derived from matrices $R_y(\omega)$, $R_z(\omega)$ that correspond to rotations by an angle ω around axes y and z of a coordinate system.

$$R_y(\omega) = \begin{bmatrix} \cos(\omega) & 0 & \sin(\omega) \\ 0 & 1 & 0 \\ -\sin(\omega) & 0 & \cos(\omega) \end{bmatrix} \qquad R_z(\omega) = \begin{bmatrix} \cos(\omega) & -\sin(\omega) & 0 \\ \sin(\omega) & \cos(\omega) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The rotation matrix $R_{zyz}(\alpha, \beta, \gamma)$ can be now expressed as

$$R_{zyz}(\alpha, \beta, \gamma) = R_z(\alpha) \cdot R_y(\beta) \cdot R_z(\gamma), \tag{4.16}$$

which gives after substitution

$$R_{zyz}(\alpha, \beta, \gamma) = \begin{bmatrix} \cos(\alpha)\cos(\beta)\cos(\gamma) - \sin(\alpha)\sin(\gamma) & -\cos(\gamma)\sin(\alpha) - \cos(\alpha)\cos(\beta)\sin(\gamma) & \cos(\alpha)\sin(\beta) \\ \cos(\alpha)\sin(\gamma) + \cos(\beta)\cos(\gamma)\sin(\alpha) & \cos(\alpha)\cos(\gamma) - \cos(\beta)\sin(\alpha)\sin(\gamma) & \sin(\alpha)\sin(\beta) \\ -\cos(\gamma)\sin(\beta) & \sin(\beta)\sin(\gamma) & \cos(\beta) \end{bmatrix}. \tag{4.17}$$

An order of matrices in equation 4.16 may seem inverse, because, according to 4.15, a matrix representing the first rotation applied, is the closest to the cloud. But it is necessary to keep in mind that all matrices $R_z(\alpha)$, $R_y(\beta)$, $R_z(\gamma)$ rotate its input around axes z, y, z and not around axes produced by previous rotations.

A formula for converting a rotation matrix $R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ to Euler angles isn't such straightforward. It is based on calculating each angle one by one from elements of the rotation matrix, starting with β. This angle is the easiest for computing, because $r_{33} = \cos(\beta)$ is directly an element of the rotation matrix. When β is known, it is possible to get other angles from the last row and column of the rotation matrix. But because of ambiguities caused by computing inverses of goniometric functions, multiple solutions can be found and it is necessary to select the valid ones by testing if the computed Euler angles match all elements of the rotation matrix.

Another method for computing Euler angles from R, presented in [19], uses a function $\mathrm{atan2}(y, x)$, which computes the $\arctan$ of the ratio $\frac{y}{x}$, but it is able to determine a quadrant of the resulting angle from signs of its two arguments y and x. With this function, two solutions (triplets of the Euler angles) can be expressed as follows

$$\begin{aligned} \alpha_1 &= \mathrm{atan2}(r_{23}, r_{13}) & \alpha_2 &= \mathrm{atan2}(-r_{23}, -r_{13}) \\ \beta_1 &= \mathrm{atan2}\left(\sqrt{r_{13}^2 + r_{23}^2}, r_{33}\right) & \beta_2 &= \mathrm{atan2}(-\sqrt{r_{13}^2 + r_{23}^2}, r_{33}) \\ \gamma_1 &= \mathrm{atan2}(r_{32}, -r_{31}) & \gamma_2 &= \mathrm{atan2}(-r_{32}, r_{31}). \end{aligned}$$

This notation is more illustrative than the rotation matrix, which makes it useful for interfacing with humans. What is more, the angles α, β, γ are not constrained by any equation, which makes generating random rotations as easy as generating tree numbers.

The main disadvantage of the Euler angles is that a special attention must be taken to preserving a consistency of the selected convention. Also, this notation is ambiguous, because it is possible to describe one rotation by more triplets of Euler angles. The next drawback is that a transformation of the Euler angles to the rotation matrix (formula 4.17) requires goniometric functions, which can be slow or inaccurate in some applications. Also, the Euler angles can't be used directly (without a thansformation to the rotation matrix) to a cloud described by a matrix. Last but not least, chaining of consecutive rotations can't be computed directly as a sum of the individual Euler angles, but some (usually more computationally demanding) process, such as transformation to rotation matrices followed by a matrix multiplication and an inverse transformation, must be used.

Quaternions

The last notation, described in [20] among others, uses a fact that each 3D rotation can be described by its axis and an angle of the rotation around this axis. A quaternion describes the rotation using four numbers as follows

$$q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} v_x \sin\frac{\omega}{2} \\ v_y \sin\frac{\omega}{2} \\ v_z \sin\frac{\omega}{2} \\ \cos\frac{\omega}{2} \end{bmatrix}, \tag{4.18}$$

where vector $\vec{v} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$ is a unit vector in a direction of axis of the rotation and ω is an angle of the rotation around $\vec{v}$.

From this definition, following constraint to elements of the quaternion can be derived

$$|q| = v_x^2 \sin^2 \frac{\omega}{2} + v_y^2 \sin^2 \frac{\omega}{2} + v_z^2 \sin^2 \frac{\omega}{2} + \cos^2 \frac{\omega}{2} = (v_x^2 + v_y^2 + v_z^2) \sin^2 \frac{\omega}{2} + \cos^2 \frac{\omega}{2} = 1. \quad (4.19)$$

Therefore q defined in this way is sometimes called a unit quaternion.

An inverse rotation q^{-1}, which "cancels out" the effect of the rotation q, can be computed from q as follows

$$q^{-1} = \begin{bmatrix} -q_0 \\ -q_1 \\ -q_2 \\ q_3 \end{bmatrix}.$$

Depending on the interpretation, it either rotates its input about angle $-\omega$ around axis $\vec{v}$ or about angle ω around axis $-\vec{v}$.

Although the definition of the quaternion can seem overcomplicated at the first glance, thanks to it the quaternions have some very beneficial properties. One of them is that a rotation matrix can be, according to 4.20, computed from them using only second order polynomials of q_i (so no goniometric function is needed, contrary to the Euler angles).

$$R = \begin{bmatrix} q_3^2 + q_0^2 - q_1^2 - q_2^2 & 2(q_0 q_1 - q_3 q_2) & 2(q_0 q_2 + q_3 q_1) \\ 2(q_0 q_1 + q_3 q_2) & q_3^2 - q_0^2 + q_1^2 - q_2^2 & 2(q_1 q_2 - q_3 q_0) \\ 2(q_0 q_2 - q_3 q_1) & 2(q_1 q_2 + q_3 q_0) & q_3^2 - q_0^2 - q_1^2 + q_2^2 \end{bmatrix} \quad (4.20)$$

Also a chaining of consecutive rotations expressed as quaternions a and b (corresponding to rotation matrices A and$B$) is possible without converting quaternions a and b to matrices. The quaternion c corresponding to rotation matrix AB can be computed according to a formula

$$c = \begin{bmatrix} a_3 b_0 + a_0 b_3 + a_1 b_2 - a_2 b_1 \\ a_3 b_1 + a_1 b_3 + a_2 b_0 - a_0 b_2 \\ a_3 b_2 + a_2 b_3 + a_0 b_1 - a_1 b_0 \\ a_3 b_3 - a_0 b_0 - a_1 b_1 - a_2 b_2 \end{bmatrix}.$$

The inverse transform, from a rotation matrix $R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ to the quaternion, can be found by computing both components, $\vec{v}$ and ω separately. The rotation angle ω can be easily evaluated from a formula

$$\frac{1}{2}(\operatorname{trace} R - 1) = \cos \omega,$$

where $\operatorname{trace} R = r_{11} + r_{22} + r_{33}$. The unnormalized rotation axis $\vec{u}$ can be estimated as follows

$$\vec{u} = \begin{cases} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix} & \text{if } R \text{ is not symmetric} \\ \text{arbitrary non-zero column of } R + I & \text{if } R \text{ is symmetric,} \end{cases}$$

where I denotes a 3×3 identity matrix. The $\vec{u}$ is not normalized, which means not only that its length doesn't need to be one, but it is also possible, that its direction is inverse to a direction of the rotation axis corresponding to computed rotation angle ω. Therefore, two unit vectors can be computed from $\vec{u}$

$$\vec{u}_1 = \frac{\vec{u}}{|\vec{u}|}, \qquad \vec{u}_2 = -\frac{\vec{u}}{|\vec{u}|}$$

and only the one which corresponds to the assigned matrix R must be selected as $\vec{v}$. The selection can be done for example by substituting of $\vec{u}_1$, $\vec{u}_2$ and ω to 4.20 and comparing the results with assigned R. Finally, when both, rotation axis $\vec{v}$ and angle ω are estimated, the quaternion can be computed using formula 4.18.

In addition to easy conversion to the rotation matrix and easy chaining of rotations, the quaternions are also relatively illustrative and not very space-demanding when stored. Their usage has also drawbacks, such as ambiguities (quaternions q and $-q$ describe the same rotation) or not very straightforward transformation of a rotation matrix to the quaternion. Also, more conventions for ordering elements in the quaternion exist.

4.2.7 Cross-correlation based orientation determination

This part of the algorithm serves for computing a cross-correlation of histograms of normals of target and source clouds, which takes place in step 1d of the pseudo-code in section 4.1. The rotation, for which this cross-correlation reaches its maximum, is very important, because it is the rotation R_{sol} from equation 2.2, searched in the whole chapter 4.2.

The cross-correlation of histograms is defined as

$$C(R) = \int_S t(\omega)\overline{s(R^{-1}\omega)}\mathrm{d}\omega, \tag{4.21}$$

where $t \in L^2(\mathbb{S}^2)$ is a continuous histogram of a target cloud and $s \in L^2(\mathbb{S}^2)$ is a continuous histogram of a source cloud. $C(R)$ is now defined as a function $SO(3) \to \mathbb{C}$, where $SO(3)$ denotes a group of 3D rotations around origin. Alike functions from $L^2(\mathbb{S}^2)$, complex square integrable functions on $SO(3)$ (denoted as $L^2(SO(3))$ can be decomposed and reconstructed too. This time, a basis consists of Wigner D-functions that will be defined further. A formula for reconstruction is

$$f(\alpha, \beta, \gamma) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \sum_{n=-l}^{l} \hat{f}^l_{m,n} D^l_{m,n}(\alpha, \beta, \gamma),$$

where $f \in L^2(SO(3))$ is a reconstructed function, $\hat{f}^l_{m,n} \in \mathbb{C}$ are $SO(3)$ Fourier coefficients and $D^l_{m,n}$ is a Wigner D-function.

Finally a main concept lying behind this algorithm can be presented. Simply put, it is possible to compute SOFT coefficients of a cross-correlation $C(R)$ by multiplying SPH coefficients of

target and source histograms. Exact formula (including a reconstruction of $C(R)$), which was derived in [10], is

$$\begin{aligned} C(R) &= \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \sum_{n=-l}^{l} \hat{t}_l^m \ \overline{\hat{s}_l^n} \ (-1)^{n-m} D^l_{-m,-n}(R) \\ &= \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \sum_{n=-l}^{l} \hat{t}_l^{-m} \ \overline{\hat{s}_l^{-n}} \ (-1)^{m-n} D^l_{m,n}(R), \end{aligned}$$

which can be also written as

$$C(R) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \sum_{n=-l}^{l} \hat{t}_l^m \ \overline{\hat{s}_l^n} \ \overline{D^l_{m,n}(R)} \tag{4.22}$$

or for band-limited $C(R)$ as

$$\boxed{C(R) = \sum_{l=0}^{l_{max}} \sum_{m=-l}^{l} \sum_{n=-l}^{l} \hat{t}_l^m \ \overline{\hat{s}_l^n} \ \overline{D^l_{m,n}(R)}.} \tag{4.23}$$

Here, $\hat{t}$ denotes SPH coefficients of a target histogram, $\hat{s}$ denotes SPH coefficients of a source histogram and D is a **Wigner D-function**. This function, evaluated in section 1(d)iii of the pseudo-code in section 4.1, is defined as

$$\boxed{D^l_{m,n}(R) = D^l_{m,n}(\alpha,\beta,\gamma) = e^{-im\alpha} d^l_{m,n}(\beta) e^{-in\gamma} = e^{-i(m\alpha+n\gamma)} d^l_{m,n}(\beta)}, \tag{4.24}$$

where α, β, γ are Euler angles corresponding to rotation R and d is a Wigner d-function. The **Wigner d-function**, evaluated in section 1(d)ii of the pseudo-code in section 4.1, is for z-y-z convention of Euler angles defined as

$$d^l_{m,n}(\beta) = \sqrt{\frac{(l+n)!(l-n)!}{(l+m)!(l-m)!}} \left(\sin\frac{\beta}{2}\right)^{n-m} \left(\cos\frac{\beta}{2}\right)^{m+n} P_{l-n}^{(n-m,n+m)}(\cos\beta), \tag{4.25}$$

where $P_a^{(b,c)}$ denotes a **Jacobi polynomial** defined (e.g. in [21]) as

$$P_a^{(b,c)}(x) = (a+b)!(a+c)! \sum_r \frac{1}{r!(a+b-r)!(c+r)!(a-r)!} \left(\frac{x-1}{2}\right)^{a-r} \left(\frac{x+1}{2}\right)^r \tag{4.26}$$

for $a, a+b, a+c, a+b+c \in \mathbb{N}_0$ and for all $r \in \mathbb{N}_0$ for which arguments of all factorials are nonnegative. Formula 4.26 is for $x = \cos\beta$ equivalent to

$$P_a^{(b,c)}(\cos\beta) = (a+b)!(a+c)! \sum_r \frac{1}{r!(a+b-r)!(c+r)!(a-r)!} \left(-\sin^2\frac{\beta}{2}\right)^{a-r} \left(\cos^2\frac{\beta}{2}\right)^r. \tag{4.27}$$

By a substitution of formula 4.27 into 4.25, we get an equation

$$d^l_{m,n}(\beta) = \sqrt{\frac{(l+n)!(l-n)!}{(l+m)!(l-m)!}} \left(\sin\frac{\beta}{2}\right)^{n-m} \left(\cos\frac{\beta}{2}\right)^{m+n} (l-m)!(l+m)! \sum_r \frac{1}{r!(l-m-r)!(n+m+r)!(l-n-r)!} \left(-\sin^2\frac{\beta}{2}\right)^{l-n-r} \left(\cos^2\frac{\beta}{2}\right)^r,$$

which can be further transformed as follows

$$d^l_{m,n}(\beta) = \sqrt{(l+m)!(l-m)!(l+n)!(l-n)!} \left(\sin\frac{\beta}{2}\right)^{n-m} \left(\cos\frac{\beta}{2}\right)^{m+n} \sum_r \frac{1}{r!(l-m-r)!(n+m+r)!(l-n-r)!} \left(-\sin^2\frac{\beta}{2}\right)^{l-n-r} \left(\cos^2\frac{\beta}{2}\right)^r$$

$$d^l_{m,n}(\beta) = \sqrt{(l+m)!(l-m)!(l+n)!(l-n)!} \sum_r \frac{(-1)^{l-n-r}}{r!(l-m-r)!(n+m+r)!(l-n-r)!} \left(\cos\frac{\beta}{2}\right)^{2r+m+n} \left(\sin\frac{\beta}{2}\right)^{2l-n-2r-m}.$$

Finally using a substitution $r = -q + l - m$, we get a complete explicit formula for Wigner d-function

$$\boxed{d^l_{m,n}(\beta) = \sqrt{(l+m)!(l-m)!(l+n)!(l-n)!} \sum_q \frac{(-1)^{m-n+q}}{(l-m-q)!q!(n-q+l)!(m-n+q)!} \left(\cos\frac{\beta}{2}\right)^{2l+n-m-2q} \left(\sin\frac{\beta}{2}\right)^{m-n+2q}.} \tag{4.28}$$

4.3 Establishing translation using FFT

In this phase of the algorithm (denoted as step 2 of the pseudo-code in section 4.1), rotational part (R_{sol} from equation 2.2) of searched transformation is known. It remains only to find a translation Q_{sol}. This task isn't as hard as finding the rotation, because we can use the already computed R_{sol}. The translation is searched in 2 consecutive steps: rough and fine.

Rough alignment

Firstly, in step 2a of the algorithm, clouds $T = \begin{bmatrix} T^x_1 & T^x_2 & T^x_3 & \dots \\ T^y_1 & T^y_2 & T^y_3 & \dots \\ T^z_1 & T^z_2 & T^z_3 & \dots \end{bmatrix}$ and $R_{sol}S = U = \begin{bmatrix} U^x_1 & U^x_2 & U^x_3 & \dots \\ U^y_1 & U^y_2 & U^y_3 & \dots \\ U^z_1 & U^z_2 & U^z_3 & \dots \end{bmatrix}$ are aligned roughly. Simply put, point T_{min} lying in a "corner" of cloud T is aligned with point U_{min} lying in a "corner" of cloud $R_{sol}S$. The points T_{min} and U_{min} are defined as

$$T_{min} = \begin{bmatrix} \min_i T^x_i \\ \min_i T^y_i \\ \min_i T^z_i \end{bmatrix}, \qquad U_{min} = \begin{bmatrix} \min_i U^x_i \\ \min_i U^y_i \\ \min_i U^z_i \end{bmatrix}. \tag{4.29}$$

The rough translation Q' can be then computed as

$$Q' = T_{min} - U_{min}. \tag{4.30}$$

4.3.1 Fine alignment

In the second step (denoted as 2b of the pseudo-code in section 4.1), fine translation Q'', which aligns clouds T and $R_{sol}S + Q'$, is found. For finding the fine translation Q'', a cross-correlation is used again. In this case, occupancy grid $t_{x,y,z}$ of target cloud T is correlated with occupancy grid $v_{x,y,z}$ of cloud $V = R_{sol}S + Q'$. The cross-correlation $(t \star s)(a, b, c)$ is, according to [22], given by the following equation

$$(t \star s)(a, b, c) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} \bar{t}_{i,j,k} \cdot s_{i+a,j+b,k+c}.$$

Similar idea as for computing a cross-correlation of functions from $L^2(\mathbb{S}^2)$ can be used here for computing a cross-correlation of occupancy grids: Fourier coefficients of the cross-correlation can be computed by a multiplication of Fourier coefficients of occupancy grids. This can be, according to [8], mathematically written as

$$\mathcal{F}(t \star s) = \overline{\mathcal{F}(t)} \cdot \mathcal{F}(s). \tag{4.31}$$

The occupancy grids can be handled as 3D discrete functions, which can be expanded using a 3D discrete Fourier transform (DFT). The 3D DFT can be computed as consecutive 1D DFT transformations along each dimmension, where the 1D DFT transformation of samples $(x_0, x_1, \ldots, x_{N-1})$ produces an N-periodic sequence of coefficients y_k defined in [23] as

$$y_k = \sum_{j=0}^{N-1} x_j e^{-2\pi i \frac{jk}{N}}, \tag{4.32}$$

where $x_k, y_k \in \mathbb{C}$ and i is the imaginary unit. Also the 3D inverse DFT can be computed as consecutive 1D transformations, defined as

$$y_k = \sum_{j=0}^{N-1} x_j e^{2\pi i \frac{jk}{N}}, \tag{4.33}$$

where $(x_0, x_1, \ldots, x_{N-1})$ are input coefficients, and $y_k \in \mathbb{C}$ is an N-periodic sequence of output samples. The 1D direct and inverse DFTs can be effectively computed using the FFT algorithm, which reduces a complexity of the computation to $N \log N$.

After computing the cross-correlation $(t \star s)(a, b, c)$ of the occupancy grids, values a_{max}, b_{max} and c_{max} for that the cross-corelation reaches its maximum are found. Transformation Q'' is computed from them as follows

$$Q'' = \begin{bmatrix} a_{max} v_x \\ b_{max} v_y \\ c_{max} v_z \end{bmatrix}, \tag{4.34}$$

where v_x, v_y and v_z are sizes of voxels in corresponding dimmensions.

When both translations, the rough one and the fine one are determined, the complete translation Q_{sol} can be finally computed as

$$Q_{sol} = Q' + Q''. \tag{4.35}$$

4.4 Modifications of the original algorithm

In order to improve performance of the original algorithm described in [8] or to overcome issues that appeared during implementation, some modifications of the algorithm were suggested. They are all described in this section.

4.4.1 Precomputing spherical harmonics and Wigner D-function

The main performance-improving modification uses a fact that in some cases, values of spherical harmonics and values of Wigner D-function can be computed before the source and target clouds are known. Concretely, values of the spherical harmonics $Y_l^m(\theta_i, \phi_i$ from step 1(c)ii of the pseudo-code in section 4.1 can be computed as soon as a maximal value l_{max} of l and positions of the centres of histogram bins are known. Because these positions are generated randomly, only their number lasts for their assignment. Therefore if the number of bins h and l_{max} are constants known before an arrival of the target and source clouds, positions of points $(H_i)_{i=1}^h = \left(\begin{bmatrix} \theta_i \\ \phi_i \end{bmatrix}\right)_{i=1}^h$ and values $Y_l^m(\theta_i, \phi_i)$ can be precomputed, which reduces a time of a registration of the target and source clouds. Additional reduction of a computational time can be achieved, instead of precomputing only $Y_l^m(\theta_i, \phi_i)$, by precomputing the whole terms $\frac{4\pi}{h}\overline{Y}_l^m(\theta_i, \phi_i)$, which can be used directly in equation 4.13.

Also, values of the Wigner D-function can be precomputed. In this case the condition is that the l_{max} and rotation samples $(R_i)_{i=1}^r$ must be known. Again, more computational time can be saved by precomputing complex-conjugated values $\overline{D}_{m,n}^l(R_i)$, which can be used directly in equation 4.23.

4.4.2 Two-level rotation determination

Although benefits of the SPH transform and the SO(3) Fourier transform are significant, a speed of the algorithm for selected resolution in rotation (density of samples R_i) can be for some applications insufficient. Therefore the "two-level rotation determination" was implemented in order to increase the speed of the algorithm and decrease its memory requirements.

This method in its first step finds a rough estimate R' of the rotation R_{sol} using a low-density rotation samples $(R'_i)_{i=1}^{r'}$ instead of $(R_i)_{i=1}^r$. Then the source scan is transformed according to the estimated rotation producing a scan $R'S$. In the second step a fine rotation R'' which

transforms scan $R'S$ towards the target scan is found. The benefit is that if the first estimate R' is correct, the second rotation is small, so rotation samples can be generated only in a surroundings of an identity rotation (represented by the identity matrix). Finally the searched rotation can be computed as $R_{sol} = R''R'$.

But this procedure can be improved even more. So far it is necessary for deternmining the R'' to find normals, create their histogram and sompute coefficients $\hat{s'}_l^m$ of cloud $R'S$. These added steps, especially finding the normals, require additional computational time which slows the registration down. Fortunately, using the implication

$$f(X) = g(R^{-1}X) \implies \hat{f}_m^l = \sum_{n=-l}^{l} \hat{g}_l^n D_{m,n}^l(R) \tag{4.36}$$

from [12], where $f, g \in L^2(\mathbb{S}^2)$ and X is a point on the unit sphere, it is possible to compute coefficients $\hat{s'}_l^m$ directly from coefficients $\hat{s}_l^m$ of the source histogram as follows

$$\hat{s'}_l^m = \sum_{n=-l}^{l} \hat{s}_l^n D_{m,n}^l(R'). \tag{4.37}$$

If the two-level rotation determination technique is used, steps 1d and 1e of the pseudo-code in section 4.1 are changed to

(d) cross-correlation $C'(R'_i)$ of the target and source histograms is computed from $\hat{t}_l^m$ and $\hat{s}_l^m$

  (i) sparse samples $(R'_i)_{i=1}^{r'} = \left(\begin{bmatrix} \alpha'_i \\ \beta'_i \\ \gamma'_i \end{bmatrix} \right)_{i=1}^{r'}$ of a space of 3D rotations around origin are selected

  (ii) values of Wigner d-function $d_{m,n}^l(\beta'_i)$ are computed for all β'_i using equation 4.28

  (iii) values of Wigner D-function $D_{m,n}^l(R'_i)$ are computed for all samples R'_i from α'_i, γ'_i and $d_{m,n}^l(\beta'_i)$, using equation 4.24

  (iv) from $\hat{t}_l^m$, $\hat{s}_l^m$ and $D_{m,n}^l(R'_i)$, a value of cross-correlation $C'(R'_i)$ is computed for all samples R'_i according to formula 4.23

(e) rotation $R' = \arg\max_{R'_i} |C'(R'_i)|$ is found as the rotation for that a magnitude of cross-correlation $C'(R'_i)$ reaches its maximum

(f) cross-correlation $C''(R''_i)$ of the target histogram and histogram of the cloud $R'S$ is computed from $\hat{t}_l^m$, $\hat{s}_l^m$ and $D_{m,n}^l(R')$

  (a) coefficients $\hat{s'}_l^m$ of the histogram of cloud $R'S$ are computed from $\hat{s}_l^m$ and $D_{m,n}^l(R')$ according to equation 4.37

(b) dense samples $(R_i'')_{i=1}^{r''} = \left(\begin{bmatrix} \alpha_i'' \\ \beta_i'' \\ \gamma_i'' \end{bmatrix} \right)_{i=1}^{r''}$ close to identity rotation are selected

(c) values of Wigner d-function $d_{m,n}^l(\beta_i'')$ are computed for all β_i'' using equation 4.28

(d) values of Wigner D-function $D_{m,n}^l(R_i'')$ are computed for all samples R_i'' from α_i'', γ_i'' and $d_{m,n}^l(\beta_i'')$, using equation 4.24

(e) from $\hat{t}_l^m$, $\hat{s'}_l^m$ and $D_{m,n}^l(R_i'')$, a value of cross-correlation $C''(R_i'')$ is computed for all samples R_i''

(g) rotation $R'' = \arg\max_{R_i''} |C''(R_i'')|$ is found as the rotation for that a magnitude of cross-correlation $C''(R_i'')$ reaches its maximum

(h) rotation R_{sol} is computed as $R_{sol} = R''R'$

The main disadvantage is that the algorithm modified in this way can skip some local optima in its first step. On the other side this technique significantly improves a speed of the algorithm and decreases its memory requirements.

4.4.3 Limiting number of voxels during translation determination

The determination of a fine translation in step 2b can be problematic if the occupancy grids $t_{x,y,z}$ and $v_{x,y,z}$ are too large. Size of voxels is always a trade-off between a computational complexity and an accuracy, because the size of a voxel corresponds to a resolution of a computed translation.

In case that the number of generated voxels is too large, a limitation that reduces a sampled area of the cloud was added to the algorithm. If a size of a cloud with assigned size of a voxel would generate more than v_{max} voxels, only v_{max} voxels is created and only the part of the cloud that lies in these voxels is used, other points are ignored.

Chapter 5

Implementation

As a part of this work, a tool for registration of point clouds with the cross-correlation algorithm described in section 4 was implemented. In this chapter, the tool is described along with other implemented utilities used for testing or processing input data.

5.1 Architecture

The cross-correlation algorithm was implemented in the C++ language and its functionality is divided into nine basic parts implemented as classes of object-oriented programming. Each of these classes is introduced here including its purpose and relations to the other classes.

5.1.1 Sph_expander

This class serves for computing coefficients of the spherical harmonic expansion of a sampled $L^2(\mathcal{S}^2)$ function. The class has three constructors, so an object of this class can be created in one of three ways. The first constructor accepts a name of the file that contains precomputed values of spherical harmonics (stored as an object Sph_expander_data described later). This way of construction is the fastest, because nothing has to be computed. The second constructor accepts a value l_{max} (maximal band of spherical harmonics) and a path to the file (in pcd format) containing pre-generated positions of centres of histogram bins $(H_i)_{i=1}^h$. In this case, values of spherical harmonic functions in points $(H_i)_{i=1}^h$ are computed during construction of the Sph_expander object. The last constructor accepts a value l_{max} and a number h of bins' centres. This constructor firstly generates h points uniformly distributed on the unit sphere (centres of bins) and then computes values of spherical harmonic functions in these points.

The most important method of this class is compute_coefficients, which computes SPH coefficients of a function from values of the function (given to the method as an input argument) and values of spherical harmonics computed in one of the constructors.

5.1.2 Sph_expander_data

This class is used as a container for precomputed values

$$\frac{4\pi}{h}\overline{Y_l^m(\theta_i,\phi_i)} \tag{5.1}$$

of (complex-conjugated and scaled) spherical harmonic functions in centres of histogram bins

$$(H_i)_{i=1}^h = \left(\begin{bmatrix}\theta_i\\ \phi_i\end{bmatrix}\right)_{i=1}^h.$$

As described in 4.4.1, these values can be used for SPH expansion of sampled $L^2(\mathbb{S}^2)$ functions, which is done by Sph_expander class.

The Sph_expander_data class stores not only values of spherical harmonics, but also coordinates θ_i and ϕ_i, for which values 5.1 were computed. An object of this class (including coordinates and precomputed values 5.1) can be saved as a file using a boost/serialization library (described in 5.4.2).

The Sph_expander_data class does not contain any logic for computing spherical harmonics. This computation is done by classes implementing the Precomputer_spharm interface.

5.1.3 Rotation_correlator

The main purpose of the Rotation_correlator class is to compute cross-correlation $C(R_i)$ of two sampled $L^2(\mathbb{S}^2)$ functions from their SPH coefficients. This task corresponds to step 1d of the pseudo-code in section 4.1. This class has two constructors, the first one accepts a path to the file that contains precomputed values of complex-conjugated Wigner D-function for rotations R_i. These precomputed values are stored as a Rot_correlator_data object escribed in 5.1.4. The second constructor accepts parameter l_{max} (specifying the maximal band of spherical harmonics used for SPH expansion) and nine other parametres specifying rotations R_i in which the cross-correlation will be evaluated. From these parameters, the constructor computes values of complex-conjugated Wigner D-function instead of loading them.

In addition to the method that computes values of the cross-correlation for all R_i, this class also contains a method for finding $R_{max} \in R_i$ for which the cross-correlation is maximal. Also a method that computes SPH coefficients of function $f(X) = g(R^{-1}X)$ only from $\hat{g}$ and R according to the right side of implication 4.36 is implemented as a part of this class.

5.1.4 Rot_correlator_data

This class serves as a container for precomputed values of complex-conjugated Wigner D-function $D^l_{m,n}(R_i)$ from step 1(d)iii of the pseudo-code in section 4.1. These values are used for computing a correlation of two histograms of normals from their SPH coefficients according to 4.23.

The Rot_correlator_data class does not store all rotations R_i. Instead, it stores the nine parametres from which the rotations can be easily reconstructed. The class (including precomputed values) can be saved as a file using boost/serialization library.

5.1.5 Translation_correlator

This class is used for translation determination from step 2 of the pseudo-code. It handles both, the rough and the fine translation. The rough one is computed directly by the Translation_correlator, the fine one is computed using the FFTW library described in 5.4.3. Unlike Rotation_correlator and Sph_expander, this class does not precompute (or load) anything in its constructor. The translation is determined by the function compute_translation, which, in addition to input clouds, accepts three arguments (each for one dimension) that determine a size of voxels used for generating occupancy grids. Also one argunent is used for limiting number of voxels in each dimension (this feature is described in section 4.4.3).

5.1.6 Precomputer_spharm

This purely virtual class represents an interface for classes computing values of scaled and complex-conjugated spherical harmonics (formula 4.5). Values computed by this function are stored in an object of the class Sph_expander_data and they are used by the Sph_expander class.

The main demand on inherited classes is that they have to implement a method precompute_spharm, which computes values of scaled and complex-conjugated spherical harmonics and stores them in an assigned Sph_expander_data object.

5.1.7 Precomputer_spharm_wolfram

This class implements the interface Precomputer_spharm (described in 5.1.6). It precomputes values of spherical harmonics using formula 4.5, which corresponds to formula (6) in [17]. That's why the class has "wolfram" in its name.

5.1.8 Precomputer_D

This purely virtual class represents an interface for classes computing values of complex-conjugated Wigner D-function. Values computed by this function are stored in an object of the class Rot_correlator_data and they are used by the Rotation_correlator class.

The main demand on inherited classes is that they have to implement a method precompute_D, which computes values of complex-conjugated Wigner D-function and stores them in an assigned Rot_correlator_data object.

5.1.9 Precomputer_D_kostelec

This class implements the interface Precomputer_D (described in 5.1.8). It precomputes values of Wigner D-function using formula 4.24, which corresponds to formula (6) in [10], hence the name of the class.

5.2 Main implemented tools

In this section, tools that were implemented in order to solve the task of registration of point clouds, are described.

5.2.1 object_search

This versatile tool, which uses above introduced classes, can serve for more purposes. The basic and simplest one is a registration of two point clouds and visualization of the result. But it is only a subset of tasks which this tool can handle, because its main purpose is a recognition of objects found in a scene. Therefore a simple distance-based clustering of the target (or more precisely scene) cloud was implemented. Also a filtering of a plane in the scene is possible, which is useful e.g. for removing a desk under scanned objects.

The recognition, i.e. selecting the best model for given target cloud (or cluster in the scene), is solved using a brute-force approach. All models are registered to the target and their similarity is computed as a value of the cross-correlation divided by the number of points of the model. The model with the highest value of similarity is then labeled as a model of the target.

All these functionalities can be adjusted in a configuration file. The default one is pre-generated, but it can be replaced by another one, which can be passed to the binary as a command-line parameter.

5.2.2 continual_localization

This tool serves for registering multiple consecutive scans stored in a directory. Each scan (except the first one) is registered to the previous one. Determined translation $Q_{sol} = \begin{bmatrix} x_{sol} \\ y_{sol} \\ z_{sol} \end{bmatrix}$ and rotation written as a quaternion $r_{sol} = \begin{bmatrix} r_x \\ r_y \\ r_z \\ r_w \end{bmatrix}$ are printed to standard output as one line $x_{sol}\ y_{sol}\ z_{sol}\ r_x\ r_y\ r_z\ r_w$ for each couple of scans.

As well as object_search, the continual_localization accepts one parameter - a path to the configuration file. Follofing parametres affect the registration:

verbose
: If this value is true, informational messages are printed to standard output.

input_dir
Directory containing pcd files that will be registered.

filter_voxel_size
Size of voxels of the voxel grid filter (used for downsampling input clouds).

normals_radius
Normal of each point is computed from the point itself and points in its neighbourhood. A radius of this neighbourhood is defined by normals_radius.

max_l
Maximal band of spherical harmonics used for SPH expansion of histograms.

sph_expander_precomputed
File with precomputed Sph_expander_data object. If this parameter is non empty, other sph_expander_* parametres are ignored.

sph_expander_sphere_pcd
Pcd file containing points uniformly distributed on a unit sphere.

sph_expander_nr_pts
Number of generated points uniformly distributed on a sphere. This parameter is ignored if sph_expander_sphere_pcd isn't empty.

rot_corr_precomputed
File with precomputed Rot_correlator_data object. If this parameter is non empty, other rot_corr_* parametres are ignored.

rot_corr_offset_deg
The smallest value of Euler angles of samples R_i.

rot_corr_step_deg
Step (in degrees) between values of Euler angles of samples R_i.

rot_corr_nr_samples
Number of samples in each Euler angle.

translation_cell_size
Size of a cell (voxel) of the Translation_correlator object.

translation_max_cells_per_dim
Maximal number of cells generated by the Translation_correlator per dimension.

5.2.3 precomp_rot_correlator, precomp_sph_expander

These tools generate, according to assigned parametres, objects of classes Rot_correlator_data or Sph_expander_data respectively. The generated objects are then serialized using boost-/serialization library and saved as a file. They can be used later for faster initialization of objects of classes Rotation_correlator or Sph_expander.

5.3 Supplementary tools

In addition to the main tools, the supplementary ones were implemented. Their purpose lies mainly in generating or preprocessing input data for the main tools. Among other smaller scripts, following tools were implemented:

5.3.1 camera

This tool displays a real-time data acquired by a connected rangefinder (MS Kinect or Xtion Pro). During the visualization it is possible to save the actually displayed cloud as a file (using the space key). The camera tool was used for acquiring most of testing data. It is very useful, because the sensor can be properly aligned to the scene and only the right scan is stored, which, comparing to common tools that store all acquired frames, decreases a processor load, saves a disc space and eliminates the time neededd for finding the correct cloud in the saved sequence.

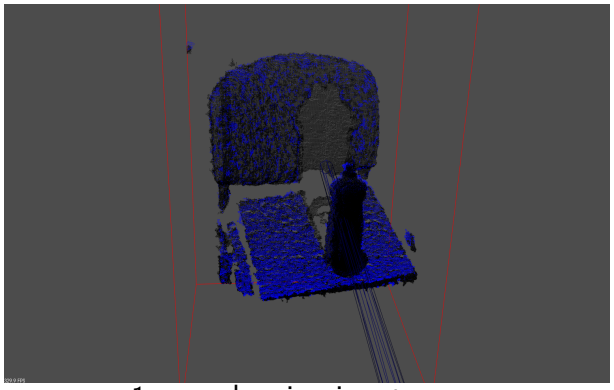
5.3.2 model_builder

Using one scanner, it is not possible to capture a 3D model of a whole object in one frame, because the scanner can see at most a half of the object at once. Therefore an interactive tool, which creates the model (one cloud) from multiple clouds (the object scanned from different angles), was implemented.

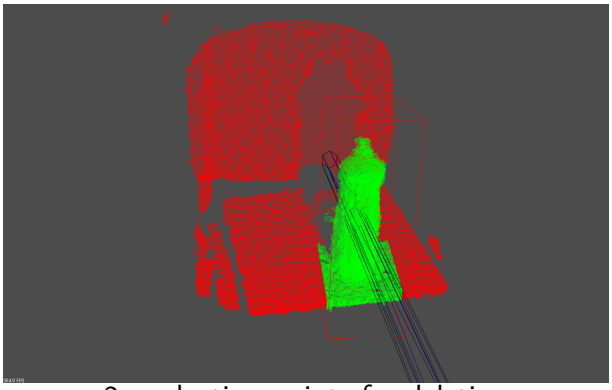
With this tool, it is possible to load all the different scans of the object, move (translate and rotate) each of them and also delete selected points. In addition to that, the model_builder is able to decimate points in selected areas and smooth the whole model by a voxel grid filter (described in [24]). But this tool doesn't take care only of points, tehir normals are handled also. So if a cloud is moved, its normals are moved as well. Thanks to this feature, all normals of the created model have a direction from the model (and not into it).

Because the model_builder generates not only the output cloud with normals, but also copies of the input clouds without deleted points and a file with executed transformations, the work can be interrupted at any time and resumed later.

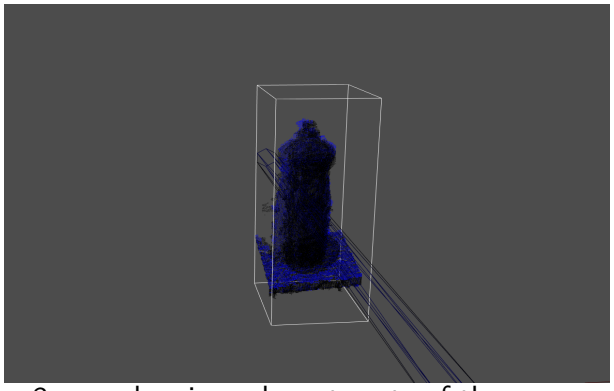
The process of building the model is illustrated by Figure 5.1.



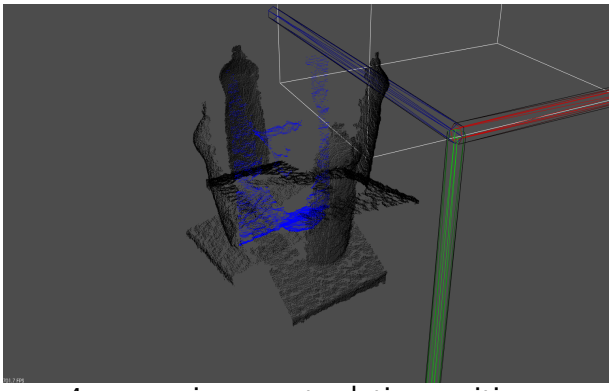
1 - overlapping input scans



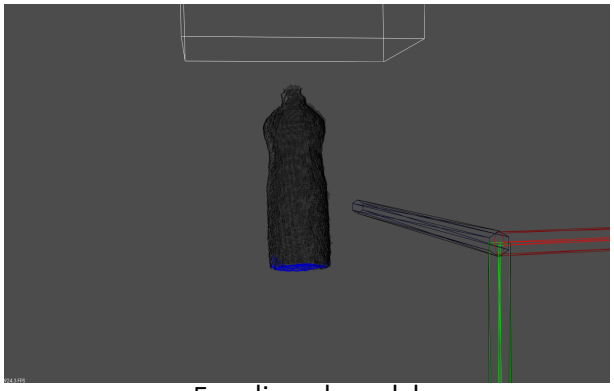
2 - selecting points for deleting



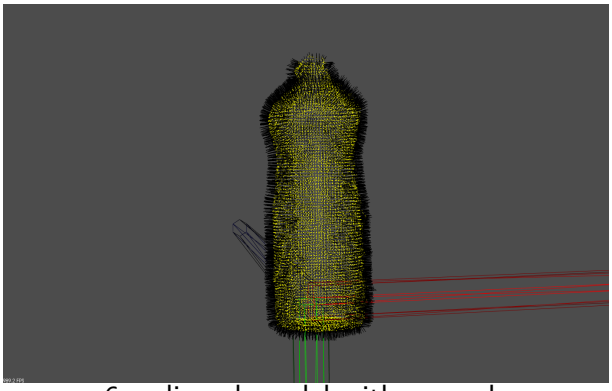
3 - overlapping relevant parts of the scans



4 - scans in correct relative positions



5 - aligned model



6 - aligned model with normals

Figure 5.1: Illustration of a process of creating a model from individual scans using the model_builder tool.

5.3.3 test_grid_generator

This tool generates a data for visual testing of the object_search tool. The test_grid_generator loads a set of objects, each described by two pcd files (one contains points, the other one contains corresponding normals) and uses a set of rotations (defined in a source code). From all these input files and the rotations, two pcd files are generated (one for points, one for normals). The generated files contain a grid composed of loaded objects (along the z-axis) rotated according to the set of rotations (along the x-axis).

An example grid produced by this tool is in Figure 5.2.

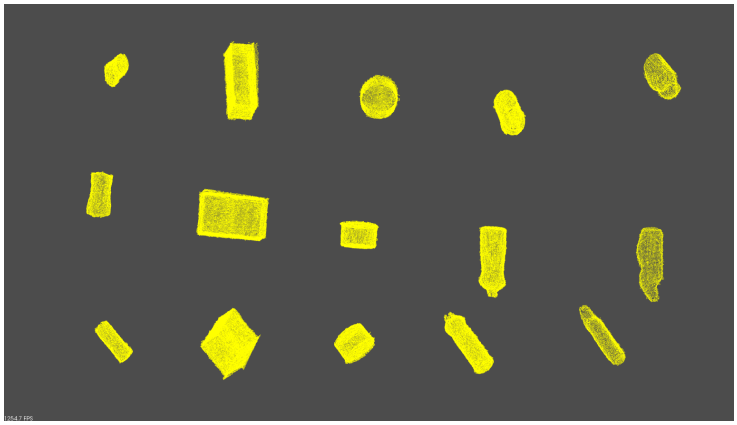


Figure 5.2: A grid generated by the test_grid_generator. It contains five object, each of them rotated by three different rotations.

5.4 Used libraries

In the implementation, sereval third-party libraries were used. The most important of them are listed in this section along with their main purpose.

5.4.1 Point Cloud Library

The following description is presented in [24]: "The Point Cloud Library (PCL) is described as a large scale, open project for 2D/3D image and point cloud processing. The PCL framework contains numerous state-of-the art algorithms including filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation. These algorithms can be used, for example, to filter outliers from noisy data, stitch 3D point clouds together, segment relevant parts of a scene, extract keypoints and compute descriptors to recognize objects in the world based on their geometric appearance, and create surfaces from point clouds and visualize them – to name a few."

This library is a base of all the implemented code, because even the class used for representing a point cloud is defined in this library. Functionalities of this library were used all over the implemented code in an effort not to duplicate once implemented features. Apart from handling points of all clouds, this library was most frequently used for acquiring data from rangefinders, transforming clouds and visualizing them.

5.4.2 Boost

The Boost project, presented in [25], is a collection of carefully-designed C++ libraries with different purposes. The following libraries from the Boost collection were used in the implemented code.

program_options
: This library was used for parsing configuration files of the implemented tools.

filesystem
: The filesystem library was used for creating and deleting files whose content was generated by implemented binaries. The library was also used for listing a content of directories and for checking an existence of files.

serialization
: This library was used for saving objects of classes Sph_expander_data and Rot_correlator_data as files (and of course for their loading later).

5.4.3 FFTW

The FFTW library is in [23] described as follows: "FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST)."

This library was used for computing the translation in section 2b of the pseudo-code in section 4.1.

5.4.4 Eigen

The Eigen library, available in [26], is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms. It was used in cases that the rotation had to be denoted in a form of a matrix or a quaternion.

Chapter 6

Experiments

6.1 Testing platform

In order to test the implemented algorithm, experiments described in this chapter were conducted. All of them were executed on a notebook HP ProBook 6550b with a core i5 processor and 4 GB RAM. An operating system used was a 64 bit Ubuntu 13.10 Saucy Salamander. For the tests, the following versions of libraries were used: PCL 1.7.1, Boost 1.53.0, Eigen 3.2.0 and FFTW 3.3.3. Executables were compiled using a gcc in version 4.4.8.1. For generating graphs, a gnuplot visualizing tool was used in version 4.6.3.

6.2 Registration on robotic dataset

This experiment examines a performance of the implemented algorithm in different configurations, processing data from the robotic dataset [1]. The dataset contains a data acquired with the MS Kinect rangefinder along with its precise (ground-truth) position which was measured by another sensors. Concretely, the first 101 scans from sequence *freiburg1_desk* of the dataset is used. This sequence contains a sweep over a table in a typical office environment (see Figure 6.1).

For the testing, from the 101 ground-truth positions of the sensor, 100 position-to-position transformations (r_{gt}, Q_{gt}) were computed (rotations are written as quaternions r). The depth data from the Kinect stored as png files were converted to clouds in a pcd file format, which can be read by the PCL library. Except the first one, each of these clouds was then registered to its predecessor, generating transformations (r_{sol}, Q_{sol}). Then errors in translation determination ΔQ were computed as an Euclidean norms of differences of $Q_{gt} = \begin{bmatrix} x_{gt} \\ y_{gt} \\ z_{gt} \end{bmatrix}$ and $Q_{sol} = \begin{bmatrix} x_{sol} \\ y_{sol} \\ z_{sol} \end{bmatrix}$ as follows

$$\Delta Q = \sqrt{(x_{sol} - x_{gt})^2 + (y_{sol} - y_{gt})^2 + (z_{sol} - z_{gt})^2}.$$

Figure 6.1: Two clouds from the freiburg1_desk sequence of the dataset [1].

Errors in rotation determination were computed using a quaternion metric described in [27]. According to it, the rotational error can be computed as

$$\Delta r = \min(|r_{sol} - r_{gt}|, |r_{sol} + r_{gt}|),$$

where $|\ldots|$ denotes the Euclidean norm (or 2-norm) of a quaternion. In addition to errors ΔQ and Δr, a time needed for the 100 registrations was measured.

Default values of configuration parameters used in the experiments are in Table 6.1. Values of parameters max_l and sph_expander_nr_pts were changed in the individual experiments, but in each of them only one parameter was modified and the other one was set to its default value.

parameter name	parameter value
verbose	false
filter_voxel_size	0.01
normals_radius	0.02
max_l	6
sph_expander_nr_pts	100
rot_corr_1_offset_deg	-6
rot_corr_1_step_deg	0.2
rot_corr_1_nr_samples	61
translation_cell_size	0.01
translation_max_cells_per_dim	100

Table 6.1: Default values of configuration parameters.

In the first experiment, a dependency of the accuracy and speed of the algrithm on maximal used band of spherical harmonics l_{max} (parameter max_l) was examined. The results are summed up in Figure 6.2 and in Table 6.2. From the graphs, a decrease of an error in rotation

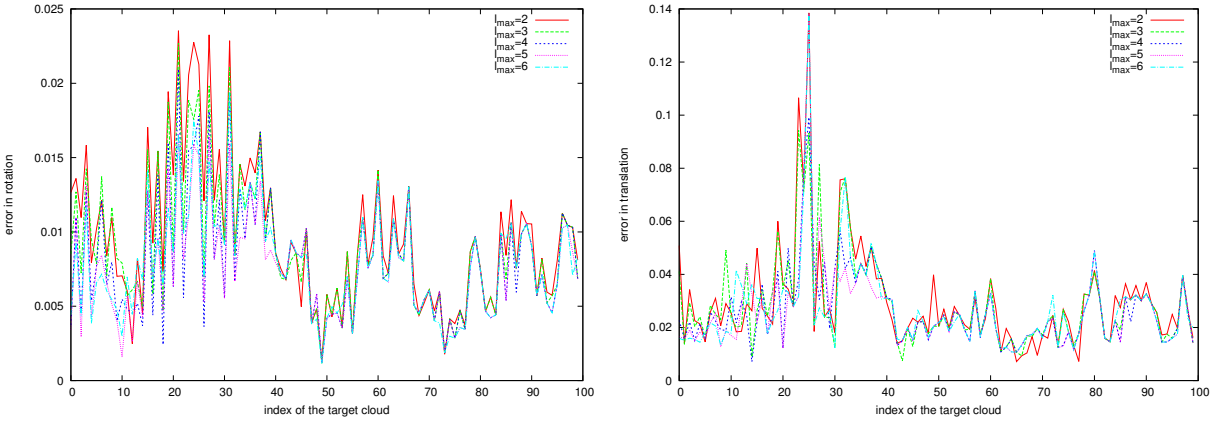


Figure 6.2: Rotational and translational error for different values of l_{max}.

is apparent with increasing l_{max}. For higher l_{max}, the input clouds are registered with higher accuracy, because histograms of their normals are represented by SPH coefficients more precisely. A drawback of increasing value of l_{max} is an increase of time needed for registration.

l_{max}	time per registration $[s]$	average error	
		rotation	translation
2	3.243	0.00969	0.0298
3	3.225	0.00895	0.0287
4	3.310	0.00804	0.0263
5	3.669	0.00761	0.0261
6	4.431	0.00769	0.0260

Table 6.2: Computation time and errors for different values of l_{max}.

The second experiment examined a performance of the algorithm depending on a number h of histogram bins $(H_i)_{i=1}^{h}$ from step 1a of the pseudo-code in section 4.1. Results of this experiment are in Figure 6.3 and Table 6.3. Especially at the beginning of the sequence, an error in rotation is smaller with higher number of histogram bins. This effect is caused by a fact that more bins can store more deatiled information about directions of normals. If the bins are too sparse, not enough details are stored in the histograms, which is a source of the errors.

h	time per registration $[s]$	average error	
		rotation	translation
30	4.271	0.00916	0.02824
50	4.206	0.00873	0.02809
80	4.235	0.00781	0.02532
120	4.298	0.00796	0.02529

Table 6.3: Computation time and errors for different numbers of histogram bins.

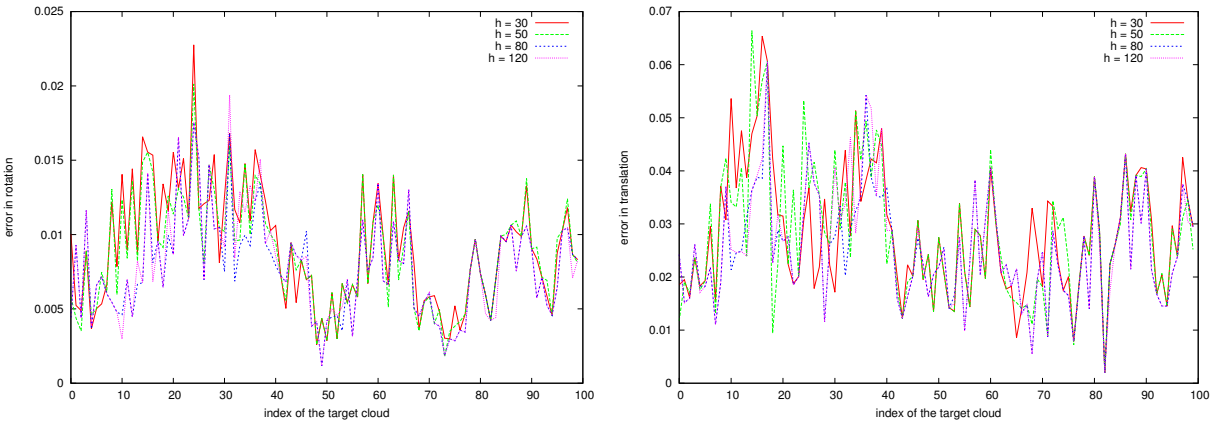


Figure 6.3: Rotational and translational error for different numbers of histogram bins.

Chapter 7

Conclusion

According to the assignment, an overview of the most popular algorithms for registration of 3D point clouds was given in chapter 3. One of them, the cross-correlation algorithm, was described in detail in chapter 4. For creating this description, it was necessary to unify a theory behind the algorithm, which had been split in different sources with different terminology.

For improving performance of the original algorithm, three modifications (precomputing spherical harmonics and Wigner D-function, two-level rotation determination and limiting number of voxels during translation determination), described in section 4.4, were suggested.

The described cross-corelation algorithm was implemented in C++ language using the Point Cloud Library. Two main tools using this algorithm were created - the *object_search*, serving for registration of couples of point clouds and the *continual_localization*, which serves for registration of more consecutive clouds. Also three supplementary applications for preprocessing data for the main tools were implemented. The first one, the *camera*, serves for real-time displaying of a data from sensor and capturing selected frame. The second one, the *model_builder*, serves for interactive generating of a full 3D model of an object from its partial scans. The third implemented supplementary tool, the *test_grid_generator*, serves for generating data for visual testing of the *object_search*.

A performance of the implemented algorithm was examined by experiments on robotic dataset [1]. The experiments described in chapter 6 involved comparison of results of the algorithm with ground-truth position of the sensor. According to these experiments, the algorithm was able to register consecutive scans in approximately three seconds with translational error in order of centimeters for selected dataset. Also an increase of the rotational error caused by decreased band of used spherical harmonics was shown. Finally, the experiments demonstrated that the error in rotation is higher for lower numbers of histogram bins.

Bibliography

[1] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proceedings of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[2] Roman Bedroš. Modul lokalizace mobilního robotu pro systém player. Bachelor's thesis, Czech Technical University in Prague, 2011.

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

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

[5] Wen L. D. Lui, Titus J. J. Tang, Tom Drummond, and Wai H. Li. Robust egomotion estimation using icp in inverse depth coordinates. In *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1671–1678, May 2012.

[6] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[7] Felix Endres, Juergen Hess, Nikolas Engelhard, Juergen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the RGB-D SLAM system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, Minnesota, 2012.

[8] Ameesh Makadia, Alexander Patterson, and Kostas Daniilidis. Fully automatic registration of 3d point clouds. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1297–1304, 2006.

[9] Eugene P. Wigner. *Gruppentheorie und ihre Anwendung auf die Quantenmechanik der Atomspektren*. Die Wissenschaft. Vieweg, 1931.

[10] Peter J. Kostelec and Daniel N. Rockmore. Ffts on the rotation group. *Journal of Fourier Analysis and Applications*, 14(2):145–179, 2008.

[11] Volker Schönefeld. Spherical harmonics, 2005.

[12] Boris Gutman, Yalin Wang, Tony Chan, Paul M. Thompson, and Arthur W. Toga. Shape registration with spherical cross correlation.

[13] Estimating surface normals in a pointcloud. `http://pointclouds.org/documentation/tutorials/normal_estimation.php`.

[14] Eric W. Weisstein. Sphere point picking. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/SpherePointPicking.html`.

[15] Eric W. Weisstein. Associated legendre polynomial. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/AssociatedLegendrePolynomial.html`.

[16] Eric W. Weisstein. Legendre polynomial. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/LegendrePolynomial.html`.

[17] Eric W. Weisstein. Spherical harmonic. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/SphericalHarmonic.html`.

[18] Eric W. Weisstein. Monte carlo integration. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/MonteCarloIntegration.html`.

[19] Lorenzo Sciavicco and Bruno Siciliano. *Modelling and Control of Robot Manipulators*. Advanced Textbooks in Control and Signal Processing. Springer London, 2000.

[20] William B. Heard. *Rigid Body Mechanics: Mathematics, Physics and Applications*. Wiley-VCH Verlag GmbH, 2008.

[21] Michael A. Morrison and Gregory A. Parker. A guide to rotations in quantum mechanics. *Australian Journal of Physics*, 40(4):465–497, July 1987.

[22] Dave Hale. An efficient method for computing local cross-correlations of multi-dimensional signals.

[23] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[24] Point cloud library. `http://www.pointclouds.org/`.

[25] Boost c++ libraries. `http://www.boost.org/`.

[26] Eigen library. `http://eigen.tuxfamily.org/`.

[27] Du Q. Huynh. Metrics for 3d rotations: Comparison and analysis. *J. Math. Imaging Vis.*, 35(2):155–164, October 2009.

Appendix

CD Content

In table 7.1 are listed names of all root directories of the attached CD with a brief description of their content.

Directory name	Description
thesis.pdf	Text of the diploma thesis.
text	Source code of the text of the diploma thesis.
c++	Source codes of the implemented tools.
tests	Scripts used for testing the implemented tools.
data	Data used for testing.

Table 7.1: CD Content