DLR-IB-RM-OP-2016-131

Local Registration by Optimizing within a Probabilistic 3D Space

Masterarbeit

Robert Lauer

Deutsches Zentrum
DLR für Luft- und Raumfahrt

Institut für Robotik und Mechatronik

BJ.: 2016

IB.Nr.: DLR-IB-RM-OP-2016-131

MASTER'S THESIS

LOCAL REGISTRATION BY OPTIMIZING WITHIN A PROBABILISTIC 3D SPACE

Freigabe:	Der Bearbeiter:	Unterschriften
	Robert Lauer	
	Betreuer:	
	Dr. Simon Kriegel	
	Der Institutsdirektor	
	Prof. Dr. Alin Albu-Schäffer	

Dieser Bericht enthält 79 Seiten, 38 Abbildungen und 4 Tabellen

Ort: Oberpfaffenhofen	Datum: 29.06.2016	Bearbeiter: Robert Lauer	Zeichen:

Local Registration by optimizing within a probabilistic 3D Space

eingereichte
MASTERARBEIT
von

Robert Lauer

geb. am 02.01.1991
wohnhaft in:
Felsennelkenanger 9
80937 München
Tel.: 0173 5830806

Deutsches Zenrum für Luft- und Raumfahrt
Dr. Simon Kriegel
Dr. Zoltan-Csaba Marton

Technische Universität München
Univ.-Prof. Dr. Martin Kleinsteuber

Beginn: 01.01.2016
Abgabe: 08.07.2016

Abstract

This thesis lays the groundwork to the development of a local registration algorithm using a 3D probabilistic data representation.
In mobile robotics the pick and place task is one of the most common. In order for a mobile robot to pick up an object it has to know its own position and orientation relatively to a given object. Since methods such as odometry and SLAM algorithms using LASER scanners are in most cases too imprecise, a different visual approach is developed in this work.
Local registration has been a research area for many years and algorithms such as the **I**terative **C**losest **P**oint (ICP) have established themselves.
The goal of this thesis is to explore the potential of a local registration algorithm, which is based on a probabilistic representation of the environment. The robot's surrounding is discretized into multiple cubes, which are also referred to as voxels. Each voxel is assigned a occupancy probability using a state-of-the-art Bayes Update. Due to this probabilistic representation various sensor scans can be used to cancel out each others uncertainties, resulting in a accurate representation of the robot's environment. Using a proven optimization algorithm a 3D model of an object is aligned with stereo camera scans of the same object seamlessly based on the occupancy probability differences of corresponding voxels.

Abbreviations

2D	Two-dimensional
3D	Three-dimensional
DLR	Deutsches Zentrum für Luft- und Raumfahrt
fps	frames per second
GPS	Global Positioning System
HDD	Hard Drive Disk
ICP	Iterative Closest Point
IMU	Inertial Measuring Unit
iiwa	Intelligent industrial work assistant
LASER	Light amplification by stimulated emission of radiation
LM	Levenberg-Marquardt algorithm
miiwa	Mobile intelligent industrial work assistant
PCL	Point Cloud Library
SLAM	Simultaneous Localization and Mapping
SLC	Small Load Carrier
TCP	Tool center point
TSDF	Truncated signed distance function

Contents

Chapter 1

Introduction

The localization of a mobile robot is an essential precondition for many of its tasks, as it makes sure that the robot's assumed position and orientation in the world coordinate system is accurate. Due to multiple uncertainties in robotic movements the expected position and orientation of a robot after a specific motion always differs from the real one. There are various tools that help minimizing said difference such as LASER scanners, GPS, IMUs, SLAM-algorithms, etc. In this thesis we use stereo cameras to acquire 3D data of the robot's environment and a workstation model, which was manually placed within a prerecorded 2D map as a basis for the registration. The previously mentioned 3D model in the 2D map was setup in the TAPAS (`www.tapas-project.eu`) and EuRoC (`www.euroc-project.eu`) EU projects' industrial robotics scenario.

Object registration is one way to determine the rigid motion between two 3D models and assumes two things: Each of the data sets represents part of the same object and both data sets overlap at least partially.

The topic of registration can be divided into two subcategories - local and global registration. While global methods estimate the rigid transformation within a global search space, local methods require an initial guess for the start configuration. The most popular algorithm in the field of local registration is the ICP (iterative closest point) [BM92], to which many variations have been developed over time. An alternative to the ICP algorithm is the NDT (normal distribution transform) [BS03] algorithm, which uses a probability based representation instead of point clouds.

This work focuses on developing a new algorithm that combines the advantages of the ICP and NDT. Thus the algorithm is supposed to be robust, which means it should find a satisfying transformation to as many scenarios as possible. One advantage the NDT has over the ICP is, that new data can be considered throughout the computation and this makes it more adaptive and robust regarding changing environments as well as sensor noise.

Figure 1.1: An exemplary small load carrier. [wur]

1.1 Problem Statement

In industrial plants a simple and repetitive type of tasks is the pick and place operation. Due to its triviality it would be desirable to automate the execution of such tasks using mobile robots. Small components such as washers or bolts are usually stored in small load carriers (SLC) as depicted in figure 1.1.
By enabling a robot to handle such SLCs, various task can be executed autonomously and this way the robot can assist the workers. SLCs are usually kept in shelves similar to the one shown in figure 1.2. The long term goal of the project we are working on is to enable a mobile manipulator to drive to a certain workstation, pick up a SLC and place it on a different workstation. As depicted in figure 1.2 we use a mobile robot and chose shelves for the workstation, as this training scenario is close to the intended real-life application. In order to fulfill this task the robot has to have precise knowledge on its position and orientation. Due to different inclinations, coefficients of friction, scattering and many more factors the goal position of the robot after a specific movement can not be predicted accurately enough and even the workstation might have a slightly different position than anticipated. The provided localization combines odometry data with LASER scanners, which recognize sections of a prerecorded map (fig. 1.4) of the working environment.
Up to now the solution for accurate localization was to use AprilTags in combination with the camera system our mobile robot is equipped with.
AprilTags are 2D barcodes developed by Edwin Olson [Ols10], which work as a visual fiducial system, employed in a variety of areas including augmented reality, robotics, and camera calibration. The software detects an AprilTag (fig. 1.3) in a provided picture and decodes the unambiguous ID, the tag's location within the picture and with a calibrated camera and the physical dimensions of the tag the relative transform between camera and tag itself. If using AprilTags the position and orientation of key locations first has to be measured and encoded into a tag and than has to be placed accurately. This solution though lacks flexibility and thus a new approach is considered.
As a further improvement we use stereo cameras to gain 3D data of the robot's

Figure 1.2: The mobile robot in front of the workstation.

Figure 1.3: An exemplary AprilTag.

environment. A 3D model of the workstation was created and manually placed within a 2D map, which was recorded beforehand using the laser scanners. The localization gained by odometry and laser data is used as an initial guess for the registration algorithm. The goal is to find the transformation that has to be applied to the camera data in order to align it with the 3D model and therefore correct our robot's position with respect to the workstation.

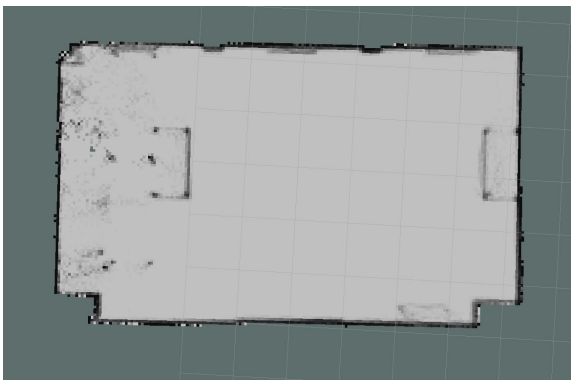


Figure 1.4: The prerecorded map, which is used by the LASER scanners for localization.

1.2 Contribution of the Thesis

This thesis presents a local registration framework, which is based on a probabilistic discretization of 3D space.
We will prove, that the approach to local registration using a probabilistic data representation instead of different data structures, such as point clouds - as used by the ICP - is a promising area of research.
The main advantage of the probabilistic approach using the Bayes update for occupancy probability computation is the fact that a quality measure of the 3D model is generated. The more sensor data is available for a certain area, the more the 3D model can be trusted and the less noise has to be considered. In order to gain a high quality environment representation multiple sensor scans can be merged together probabilistically.
Different representations of optimization parameters encoding translations and rotations are tested on matrices in combination with the Levenberg-Marquardt algorithm.
These representations are paired with a variety of cost functions in order to find the best pair for optimization with the Levenberg-Marquardt approach.
The algorithm generates a discretized probabilistic representation of the 3D model of an object and of the corresponding stereo camera data. By comparing occupancy probabilities of interrelated areas the orientation of both data sets is rated and improved using the before mentioned Levenberg-Marquardt optimizer.

The fact, that the chosen cost function assumes its minimal value in the case of seamless alignment is proved with several test scenarios including translations in x-, y-, and z-direction as well as rotations around the x-, y-, and z-axis.

1.3 Thesis Outline

Chapter 2 will introduce the reader to the state of the art in the area of research covered by this thesis. Existing and commonly used algorithms will be presented after discussing what our robot is already capable of and which problems still need to be solved. In the end the preliminary work that has been done as preparation for this thesis will be explained. Chapter 3 introduces the reader to the basics behind the newly developed algorithm. We explain how the 3D model was generated and how the camera data was acquired, before focusing on the used data structure. Afterwards an introduction to the generation of our probabilistic voxel spaces and the computation of occupancy probabilities is described. A short explanation on how to compute the probabilistic voxel spaces that represent our camera data and our 3D model is also given. In the next chapter we focus on optimization. The two considered algorithms by Levenberg-Marquardt and Nelder-Mead are presented in detail before we show different representations for rotations, that affect our optimizer. The three tested cost functions our optimizer was supposed to use are then introduced and their advantages and disadvantages are discussed. In the end of chapter 4 we show and explain the compromise we chose as representation for the transformation and the final cost function. Chapter 5 then concludes this work presenting our positive results, proving the feasibility of our approach and in the end a summary of research topics for future work is given.

Chapter 2

Local Registration - State of the Art

In order to execute simple pick and place tasks autonomously we employ a mobile robot equipped with a light weight robot arm, a pan-tilt unit (PTU) and a multitude of sensors (see Fig 1.2). The fundamental abilities the mobile manipulator needs in order to carry out an exemplary task fully autonomously were presented in [DKBS15]. Based on a 2D map, which has to be provided or previously recorded, the robot has the ability to navigate from its current position to the workstation and vice versa. Due to the previously mentioned factors of uncertainty, we can not simply teach the exact position of the shelves. Thus, the mobile manipulator has to use its sensors to find the correct position in front of the work station. As previously explained, the approach by [DKBS15] was to use AprilTags as reference. Due to the limits of flexibility and the high expenditure we searched for an alternative. Local registration seemed to be a promising solution and therefore, the ICP and NDT algorithms were considered. After finding that the NDT is a legitimate alternative to the ICP we developed the idea of our own probabilistic approach to local registration.

2.1 Iterative Closest Point (ICP)

In 1992, Besl and McKay introduced the esteemed ICP algorithm [BM92]. Then again Rusinkiewicz and Levoy present a summary of variations in 2001 [RL01], which are aiming at accelerating the ICP algorithm. In this work, the ICP will be used for the minimization of the pose error and fine matching of a prerecorded model with online obtained camera data from the object in different poses - a process, also denoted as 3D registration.
As depicted in fig. 2.1 we assume to have two fairly similar point clouds A and B respectively consisting of n points a_i ($i = 1, \ldots, n;\ n \in \mathbb{N}$) and m points b_j ($j = 1, \ldots, m;\ m \in \mathbb{N}$). The goal of the ICP algorithm is to find the transformation matrix, which has to be applied to point cloud A in order to minimize the difference

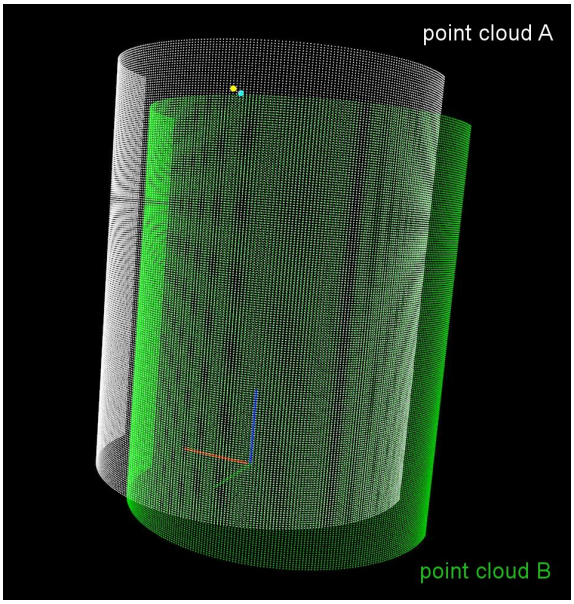


Figure 2.1: An example of two point clouds the ICP algorithm could align.

towards point cloud B. Transformation matrices contain a 3×3 rotation matrix R and a 3×1 translation vector t, which results in the following appearance:

$$M = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

The difference between the point clouds is measured as follows: For each point a_i $(i = 1, ..., n)$ in point cloud A, the nearest neighbor b_j $(j = 1, ..., m)$ in point cloud B, within a predefined search radius, is found and the distance d_i between them is calculated. The search radius around a_i, within which b_j is searched for, is one of the adjustable parameters of the algorithm, others are maximum number of iterations, accepted total difference between both point clouds, etc. The sum of all these distances $d_{sum} = \sum_{i=1}^{n} d_i^2$ serves as an error measure. By applying different transformation matrices to point cloud A, the ICP iteratively uses a numerical approach to minimize d_{sum}. The algorithm continues the search until it either converges (the accepted tolerance is achieved) or the maximum amount of iterations has been reached.

2.2 Normal Distributions Transform (NDT)

In 2003 Biber and Straßer introduced the NDT [BS03] as a new approach for laser scan matching. The main difference to previous approaches is that instead of finding correspondences between two scans in order to match them, it uses the normal distributions transform as an alternative representation of the data. The NDT locally models the probability of measuring a point. The result of the transform is

a piece-wise continuous and differentiable probability density, that can be used to match another scan using Newton's algorithm.
We use the Normal Distributions Transform (NDT) algorithm - just like the ICP - to determine a rigid transformation between the model point cloud and the 3D point cloud acquired by our stereo cameras.
In the following, the NDT is briefly explained based on the explanation of [SAS+13]. For simplicity reasons we, too, chose to limit the explanation to the 2D case (3D is straightforward).
By placing a local normal distribution in the position of every 2D-points of one scan, the NDT models its probability distribution. The two-dimensional space, in which the data is expected is divided into cells with constant size. If a cell contains three or more points, proceed as follows:

1. Identify every 2D-point $x_{i=1,\dots,n}$ in the current cell

2. Calculate the mean of all 2D-points inside each cell $q = \frac{1}{n}\sum_i x_i$

3. Calculate the cells covariance matrix $\Sigma = \frac{1}{n}\sum_i (x_i - q)(x_i - q)^\intercal$

The probability of measuring a sample at 2D-point x contained in this cell is now modeled by the normal distribution $N(q, \Sigma)$:

$$p(x) \sim exp(-\frac{(x-q)^\intercal \Sigma^{-1}(x-q)}{2}) \tag{2.1}$$

With the NDT we get the probability to measure a sample at any position within the 2D space our cells cover. This results in the probability density for our 2D space, which is an analytic piece-wise continuous and differentiable function.

2.3 Case Studies for Given Local Registration Methods

The ICP algorithm is the most common in local registration and the NDT's importance and popularity is consistently growing. Both seemed to be a viable option and thus, were tested. In the preliminary work of this thesis a simple comparison between ICP and NDT was made. A test set of data was recorded in front of the work station using the stereo cameras and the 3D model of the shelves was placed in its expected position. Each shot was then registered to the 3D model using the ICP algorithm and the NDT algorithm. The robot has to stand close to the shelves, because the length of its arm is limited, which has an effect on the position of the stereo cameras. As a result each single scan only catches a part of the shelves, due to the limited field of view of the cameras.

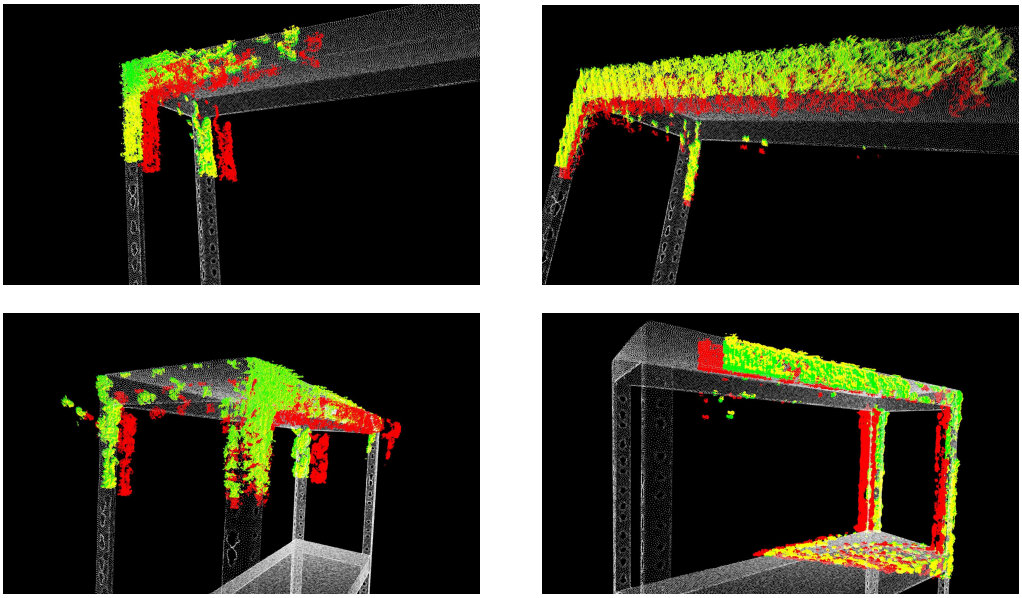


Figure 2.2: GRAY=3D model, RED=initial stereo-camera data, YELLOW=result of ICP, GREEN=result of NDT; A visual comparison of the results the ICP and NDT produce. In most cases the results of ICP and NDT are almost identical.

In most cases both algorithms gave a satisfying result, which means that a transformation was found that improved the alignment of the camera data with the 3D model significantly. We found both algorithms to perform well if the right set of parameters is found. It is hard to find a set, which works well for all shots, because the distance between initial guess and real position can vary quite a bit between the different views. Figure 2.2 shows 4 exemplary cases, where the ICP and NDT produced acceptable results.
The comparison of both algorithms was visually made by simply comparing the resulting position of the ICP algorithm, the NDT algorithm, the original position and the 3D model's position. As a summary we can say that both algorithms worked fine in most cases but the ICP as well as the NDT had some constellations they were not able to solve. Figure 2.3 shows an exemplary case where the ICP did not manage to improve the position and figure 2.4 shows the counterpart from the NDT.
One of the most significant drawbacks was the time both algorithms needed to find a acceptable solution and the fact, that both failed in almost 20% of the test cases. This resulted in us starting to develop a new algorithm that combines the advantages of the ICP and the NDT algorithm. The forte of the ICP is that the years of development resulted in a very fast performance that works in many applications without much adaptation. The NDT on the other hand has the advantage that the result improves with more and more provided measurements, because it can - due to its probabilistic representation - update its solution. The long term goal is to get an algorithm that is faster and more robust than the previously tested approaches.

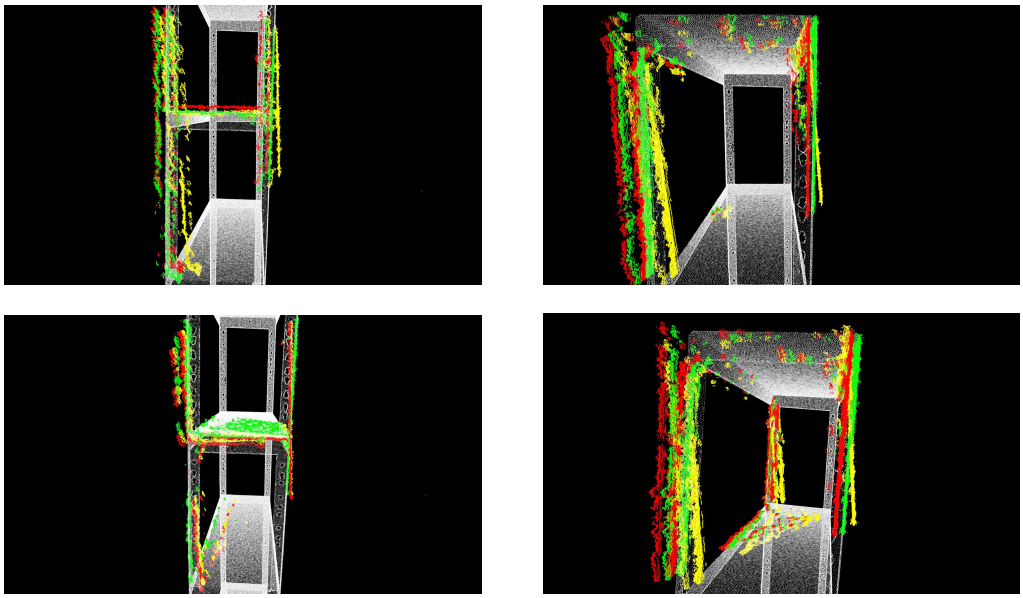


Figure 2.3: GRAY=template, RED=initial stereo-camera data, YELLOW=result of ICP, GREEN=result of NDT; A visual comparison of the results the ICP and NDT produce. In some cases the ICP applies a rotation, which makes the result worse.

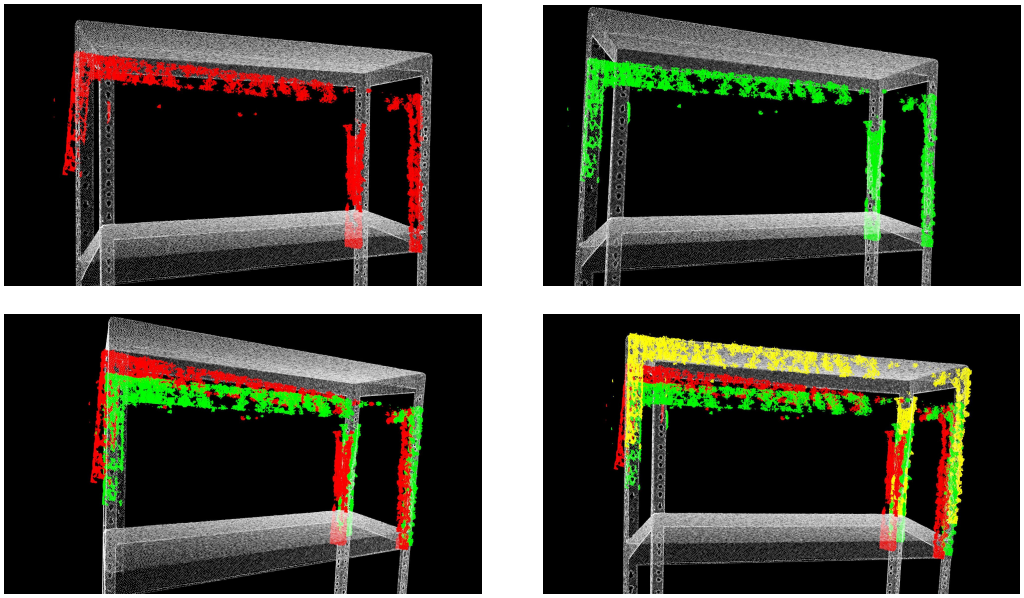


Figure 2.4: GRAY=template, RED=initial stereo-camera data, YELLOW=result of ICP, GREEN=result of NDT; A visual comparison of the results the ICP and NDT produce. In one case the NDT did not apply a translation upwards, which made the result worse.

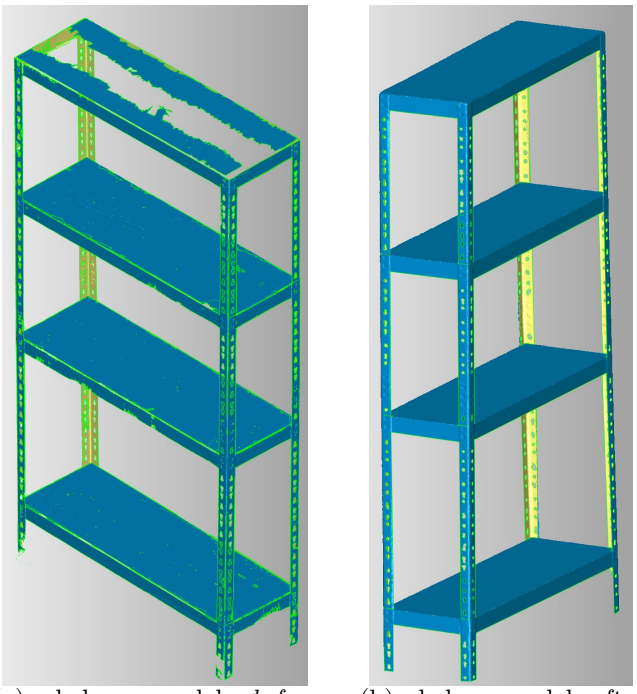
Chapter 3

Basics

The problem of consistently aligning two or more point clouds is known as registration. The goal is to find the relative orientation and position of the separately acquired point clouds in a global coordinate system, in such a way that the overlapping areas merge seamlessly.
There are two different types of registration, which have to be distinguished in this context - global and local registration. If the given transformation between two point clouds is small and a relevant overlap between them can be assumed, local registration is used. In case the point clouds are not overlapping and the transformation is large, global registration is the most common method.
In our case a 2D map of the working area was recorded by the laser scanners mounted on the miiwa robot using a variation of the SLAM algorithm. Additionally a 3D model of the workstation was created using a laser scanner to reach maximum accuracy. This model was placed as precisely as manually possible into the correct position within the 2D map.
While depth images of different views of the workstation were recorded, the odometry and the laser scanners were used to create an initial guess of the robot's position and orientation at the time.
Thus, we have a previously recorded model of a workstation and aligned point clouds acquired from different views with our pan-tilt cameras to improve that initial guess regarding the localization. Therefore, a system that is able to find the transformation matrix, which has to be applied to our camera data in order to align it with our model, is needed. This correction of the initial guess is mandatory for accurate 3D modeling due to the uncertainties of the robot's movement and calibration inaccuracies.

3.1 Groundtruth

The two point clouds our local registration algorithms register towards each other are a previously recorded model (fig. 3.1(a)) and a depth image, which is calculated from the data the stereo cameras collect. A Faro Platinum measurement arm with



(a) shelves model *before* post-processing

(b) shelves model *after* post-processing

Figure 3.1: A comparison between the aligned model before post-processing (left) and after post-processing (right).

a Nikon ModelMaker D (fig. 3.2) and the Focus Handheld software were used to create the 3D ground truth model of the shelves (fig. 3.1(b)). Due to the limited workspace of the arm and the size of the shelves the front and the back side had to be scanned separately. For aligning and uniting both scans as well as post-processing such as smoothing, filling wholes, etc. the GeoMagic software was used.

Due to the fact that the front and back of the model had to be combined after scanning and the post-processing, a significant amount of accuracy was lost. The initial idea behind scanning the shelves using a laser scanner was to get a perfect model of our work station. The model mesh consists - even after downsampling - of about one million triangles, which makes the file very big and this results in high processing times. A better solution would be a manually generated 3D model, which can be produced with tools like blender or similar software. This way our 3D model could be reduced to about 0.1% of its current data size and thus, computations could be accelerated significantly.
A global 2D map of the lab, which contains our work station, was created with software - similar to SLAM - provided by KUKA using the laser scanners affixed to our mobile robot, the KUKA miiwa (fig. 1.4). Once the robot localized itself within the aforementioned map, we positioned it in such a way that the shelves were within

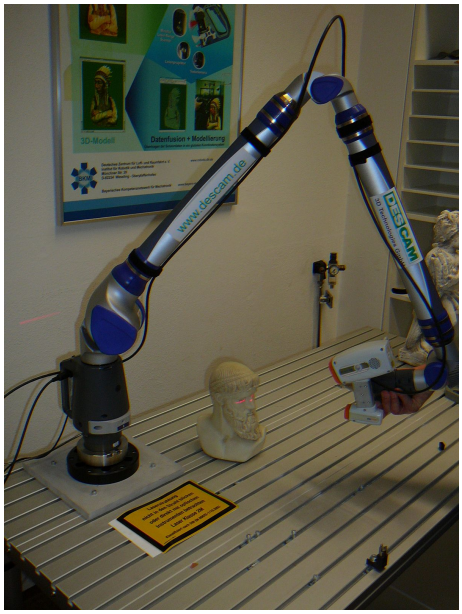


Figure 3.2: The Faro Platinum measurement arm in combination with a Nikon ModelMaker D used to create 3D-models.

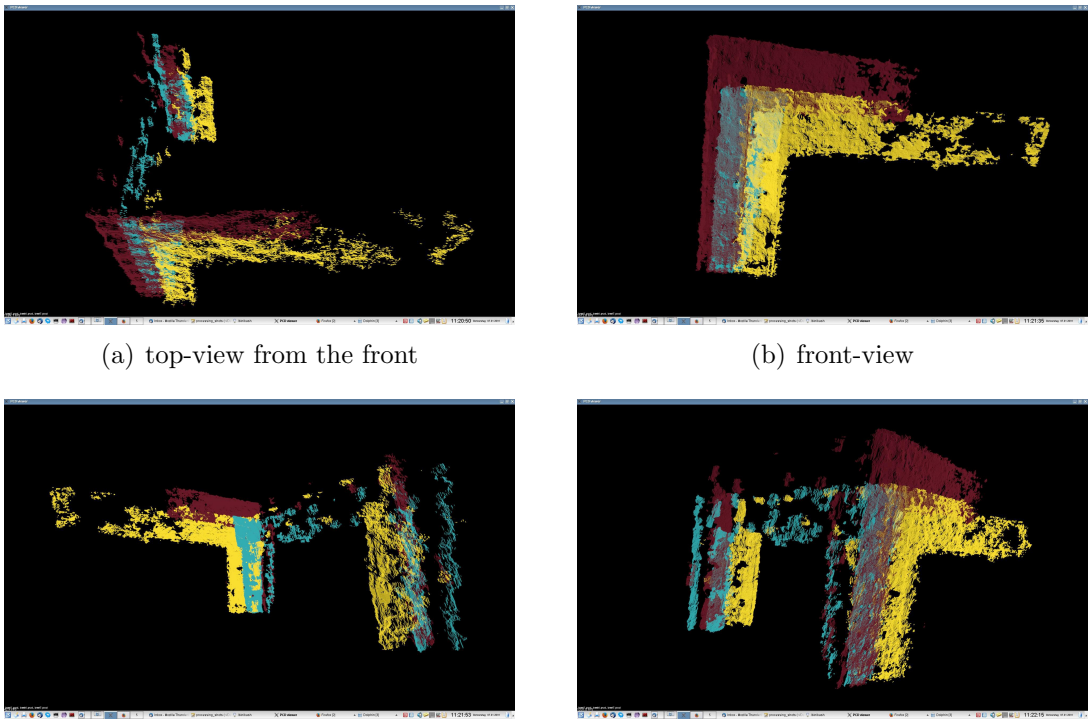
the field of view of the stereo cameras mounted on the pan-til unit. Based on this view the position of the shelves 3D model within the 2D map were manually places as precisely as possible.

3.2 Data Acquisition

The general idea to improve the localization of the mobile robot was to use local registration algorithms to register live stereo camera data to a prerecorded model of the workstation. 27 shots of the shelves were taken, with 3 different tilt angles (horizontal, $17.19°/0.3rad$ above, $28.65°/0.5rad$ below) and 9 different pan angles ($11.46°/0.2rad$ steps). We expected the movement of the pan-tilt unit to be very precise and because of that the overlapping parts between neighboring shots should have aligned almost seamlessly. Figure 3.3 shows an example of how badly the neighboring shots really align.

The error between the separate shots is an accumulation of different aspects such as inaccuracy of the pan-tilt unit and errors in the transformation from the camera coordinate system to the global system. We tried to minimize the effects of those sources of error but were not able to get a better result than shown above.

Because of this inaccuracy we were not able to simply unite all the shots into one big point cloud and register it to the prerecorded model, as initially planned. The preliminary procedure is to register every shot separately to the template and evaluate the developed algorithm based on this information.



(a) top-view from the front

(b) front-view

(c) back-view

(d) front-view from the left

Figure 3.3: An example of the real alignment of three neighboring shots from different camera positions, which overlap significantly but are poorly aligned.

3.3 Data Structure: Octrees and Their Usage

The data a sensor generates while perceiving its environment has to be stored in and represented with a certain data structure. Wurm et. al. present some of the possibilities in their paper [WHB+10]. When working with stereo-cameras the most obvious structure would be a point cloud. Nonetheless, possible and widely used alternatives are elevation maps [Heb89], multi-level surface maps [TPB06] and 3D maps based on octrees, called Octomaps [WHB+10], which are depicted in figure 3.5). Octomaps use Octrees to divide the space that is to be represented into cubes of arbitrary size. These cubes are also referred to as volumetric elements or voxels for short (derived from pixels). Each voxel is assigned a occupancy probability and a corresponding state of occupancy of the space it contains, which is why they can have one of three states ("free", "unknown", "occupied"). Octrees are tree structures with either 8 successors or children per node or none. They are a derivation from binary trees and Quadtrees. Binary trees are convenient for the representation of 1D data, Quadtrees for 2D data and Octrees for 3D data. A generalization to data of arbitrary dimensions is referred to as N-Tree. The basic idea behind Octrees is that parent nodes are subdivided into 8 subnodes, if the state of one of those children nodes differs from the state of their parent and thus their siblings. If on the other hand, the state of the eight children nodes would be the same, these are combined into their parent node and no further subdivisions in this branch will be performed. The main field of applications of Octrees is the area of computer graphics. 3D data is hierarchically subdivided in such a way that the parent node contains all the data of its children nodes, the children nodes on the other hand contain one eighth of the parent node's data. This makes Octrees a proper data structure for divide-and-conquer-strategies. The procedure of subdividing a space is depicted in figure 3.6. The whole space is fitted into one root cube, which is then subdivided into 8 sub-cubes, which then again are also subdivided into 8 sub-cubes respectively, and so on, until the highest desired resolution is reached. This structure enables Octrees to provide fast operations for both insertion and query of data, while keeping memory consumption low.
The data structure we use in this thesis - called DynamOctree - was developed at the DLR and is also based on Octrees, which makes it similar to Octomaps. A DynamOctree contains multiple Octrees, while each Octree is unambiguously assigned to a position in a grid. The general concept is visualized in figure 3.4 from [Kri15].
Each Octree consists of a C++-Map and contains additional information such as ID, origin, resolution and size. This structure provides the possibility of arbitrarily extending the covered space into any direction or dynamically pushing individual Octrees to the HDD in order to be more memory efficient.

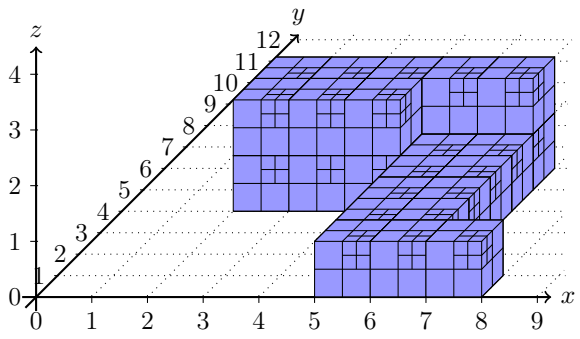
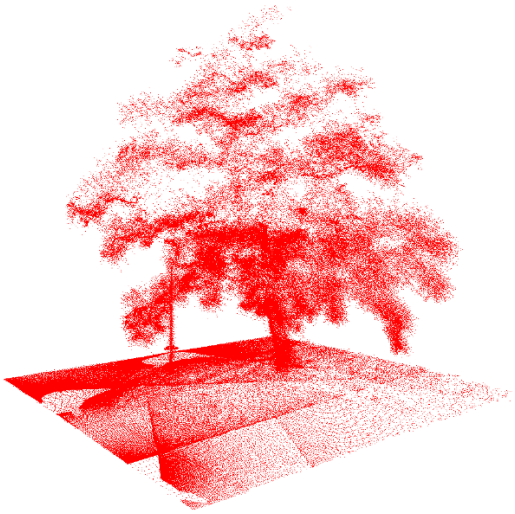
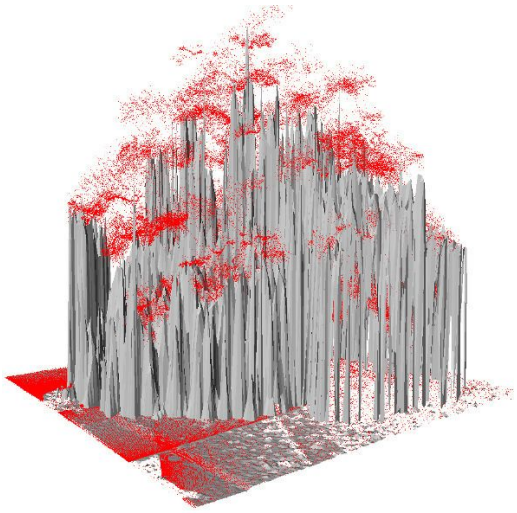


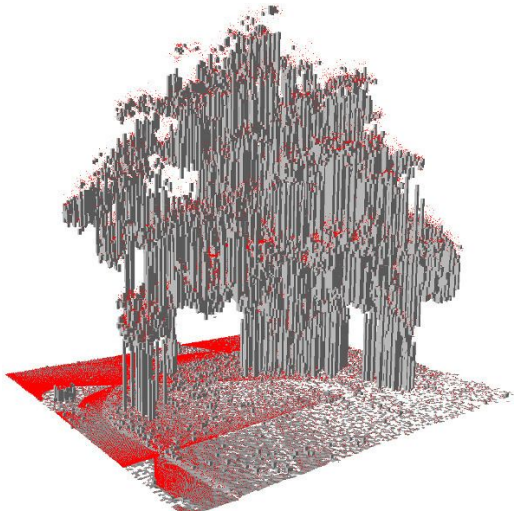
Figure 3.4: Multiple Octrees united into one DynamOctree



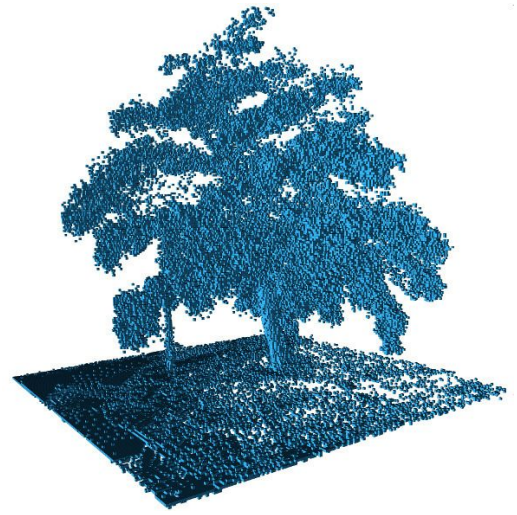
(a) point cloud



(b) elevation map



(c) multi level surface map



(d) Octomap

Figure 3.5: A visualization of the most common 3D data structures presented by [WHB+10]

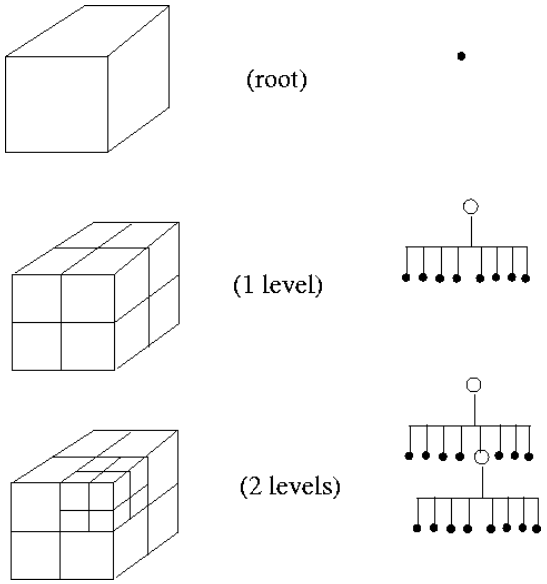


Figure 3.6: Three levels within an octree represented as cube and tree. [For]

Chapter 4

Local Registration Based on a 3D Probabilistic Space

The approach presented in this thesis is quite complex and consists of multiple steps. In the following we will describe the general idea and procedure of the local registration based on a 3D probabilistic space and explain the most important individual steps in detail afterwards.

4.1 General Procedure

As previously explained, the basic idea is to use DynamOctrees to represent the camera data as well as the 3D model of our workstation in order to register both data sets towards each other. A visualization of the individual steps is depicted in figure 4.1. The first thing our algorithm has to do, is to load the 3D data it needs. The shots our stereo cameras took are saved in the form of depth images to our HDD. One arbitrary view is loaded into a data structure called DepthImageStorage, that was developed at the DLR. The other data set our algorithm need is the 3D model of the shelves. This is stored as a triangle mesh on our HDD and is loaded into a data structure called TriangleStorage.

> **Triangle Storage [Kri15]:** For efficiency reasons such as reduction of memory consumption and improved traversal of the edges, the triangle faces are not stored explicitly. An alternative representation where a triangle mesh is denoted as:
>
> $$\mathcal{M} := (\mathcal{V}_\mathcal{M}, \mathcal{E}_\mathcal{M}),$$
>
> is introduced. A triangle mesh consists of n vertices $\mathcal{V}_\mathcal{M}$ and m directed edges $\mathcal{E}_\mathcal{M}$, which are defined as follows:

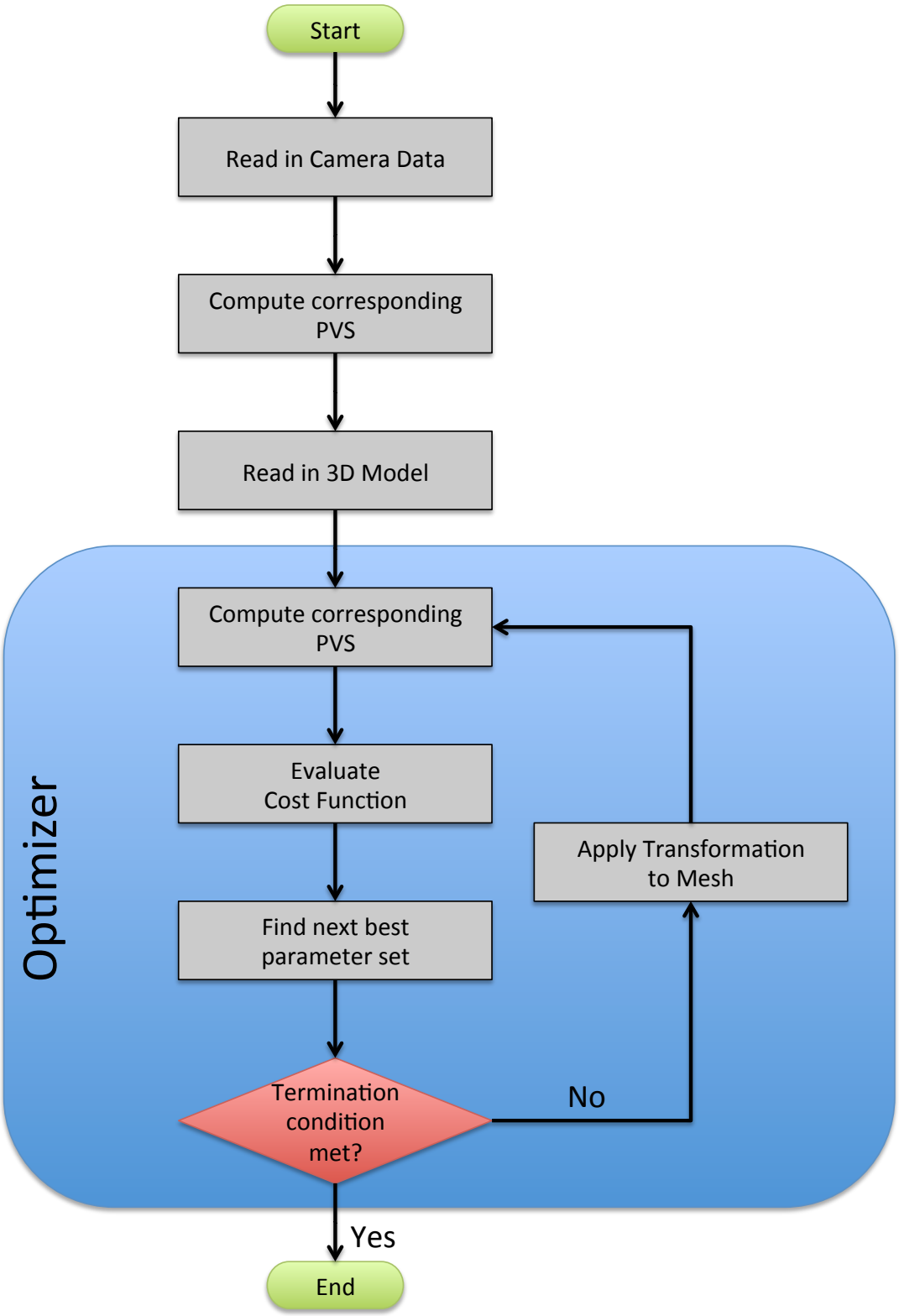


Figure 4.1: A visualization of the general procedure the newly developed algorithm follows.

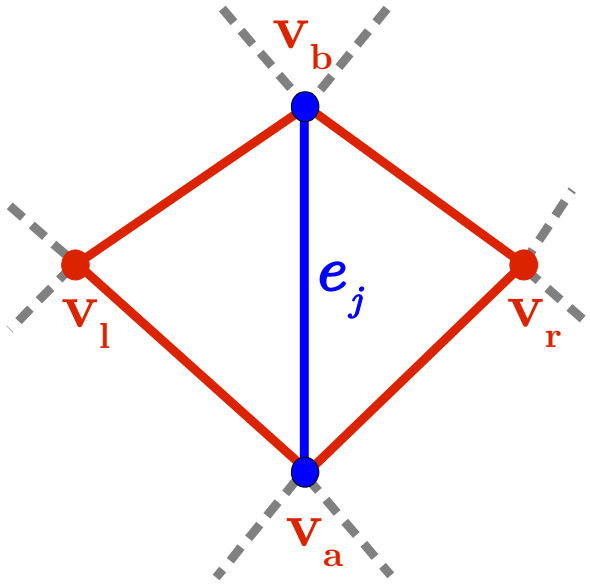


Figure 4.2: Each edge e_j of a trianlge mesh consists of two vertices v_a and v_b. Additionally the two closest vertices to e_j are stored, as they build the two adjacent triangles. [Kri15]

$$\mathcal{V}_\mathcal{M} := \{v_1, ..., v_n\} \in \mathbb{R}^3$$
$$\mathcal{E}_\mathcal{M} := \{e_1, ..., e_m\}$$

Two adjacent vertices v_a and v_b are connected by an edge e_j with the direction:

$$e_j = dir(v_a, v_b) \text{ with}$$
$$dir(a, b) = \frac{b - a}{|b - a|}$$

Additionally the neighboring vertices v_l and v_r are stored, because they are the missing part to the adjacent triangles on both sides of the edge e_j. At last the corresponding surface normal n_i is calculated and saved for each vertex v_i.

We want to work with a probabilistic representation for our data, because this lets us consider new data during registration and reduces the influence of noise. For this purpose both data sets have to be converted into Probabilistic Voxel Spaces (PVSs) using the Bayes Update. Two empty voxel spaces with identical origin, orientation, resolution and size are initialized, while each voxel is assigned the probability 0.5 ("unknown").

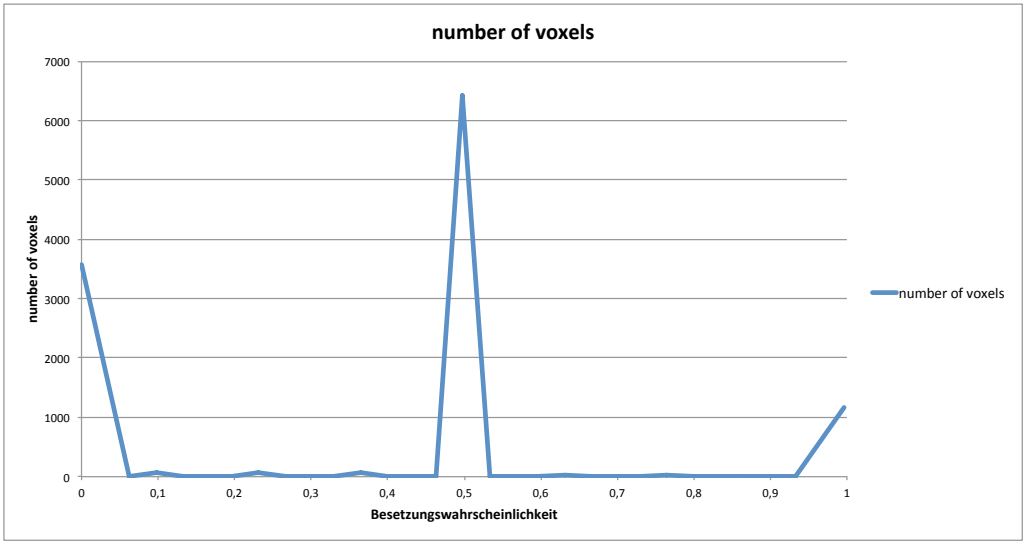


Figure 4.3: A typical distribution of voxel probabilities within the PVS representing the camera data.

For the computation of the probabilities of the PVS representing the camera data, each beam is checked in terms of independence regarding previously considered measurements. If a beam is accepted (considered independent), the probability of each penetrated voxel is updated according to our sensor model and Bayes' theorem. Multiple factors, such as sensor uncertainty due to increasing distance, different surface properties, etc. are considered. A more detailed explanation of the probability computation is given in chapter 4.2. A typical distribution of the probabilities assigned to the voxels can be seen in figure 4.3.
The voxels of the PVS representing the mesh can only assume two values, either 0 ("free") or 1 ("occupied"). Each voxel that is intersected by an edge is set to "occupied" and all others are assumed to be free. This approach is founded on the fact, that our 3D model is supposed to be a perfect copy of the real shelves and thus, no noise can occur.
The task of improving the registration between both PVSs is taken by a optimization algorithm. It iteratively tries different transformations and evaluates, whether they improve the result based on a predefined cost function. The optimization algorithm can adjust a parameter set, which encodes a transformation. Different rotations and translations are applied to the 3D model of our shelves and the transformed mesh is then again converted into a PVS. The rating on whether a applied transformation improved the position of both data sets with respect to each other is evaluated based on a certain cost function. The goal for the optimizer is to minimize the difference between corresponding pairs of voxels from both PVSs. Due to the fact, that both PVSs have the same origin, orientation, resolution and size the probability difference should be at a minimum, if both data sets are aligned perfectly. In this case every voxel from the camera PVS that corresponds to an "occupied" voxel in

the model PVS should have a probability close to 1 and every voxel from the camera PVS that corresponds to a "free" voxel in the model PVS should have a probability close to 0. The optimizer tests various transformations in a loop and does not stop until one of its termination conditions is met. The resulting transformation should consist of the rotation and translation our mobile robot has to perform in order to correct its position and orientation regarding its workstation. The main advantage over the ICP algorithm is that the computed probabilities can be updated with additional measurements. This means that by probabilistically merging multiple shots of the same part of an object noise can be reduced, resulting in a more accurate representation.

4.2 Computation of Voxel Probabilities Using Bayes' Theorem

By creating a probabilistic voxel space we discretize the 3D space regularly. Each voxel is assigned a real value $p(x)$ corresponding to the occupancy probability or the amount of matter inside its volume. As the occupancy probability of the space is unknown, as long as no measurement is considered, all voxels are initialized with the probability 0.5. Based on this initial state the probability is updated according to Bayes' theorem for each measurement:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Bayes' theorem relates the conditional probability $P(A|B)$ to its probabilistic counterpart $P(B|A)$, by saying that the probability of occurrence of an event A given the event B $P(A|B)$ can be calculated from the probability of an event B given the event A $P(B|A)$ and the prior probabilities of the individual events A $P(A)$ and B $P(B)$. By applying this theorem to the update problem of our PVS we get:

$$P(occ_i|meas) = \frac{P(meas|occ_{i-1}) \cdot P(occ_{i-1})}{P(meas)}, \tag{4.1}$$

where $P(occ_{i-1})$ represents the current occupancy probability for a voxel and $P(meas)$ represents the probability for a certain measurement, which are both a-priori-probabilities. It is hard to calculate $P(meas)$, because all possible outcomes of a measurement would have to be considered. A possibility to avoid said problem is to consider the probability of vacancy $P(free)$. The corresponding formulation of Bayes' theorem would be:

$$P(free_i|meas) = \frac{P(meas|free_{i-1}) \cdot P(free_{i-1})}{P(meas)}, \tag{4.2}$$

By solving equation 4.2 for $P(meas)$ and adding it to equation 4.1 we get a formula, which does not require the calculation of $P(meas)$:

$$\frac{P(occ_i|meas)}{P(free_i|meas)} = \frac{P(meas|occ_{i-1}) \cdot P(occ_{i-1})}{P(meas|free_{i-1}) \cdot P(free_{i-1})} \tag{4.3}$$

Assuming independent measurements we can simplify 4.3 get the so-called odds-form of Bayes' theorem:

$$\Leftrightarrow \frac{P(occ_i|meas)}{P(free_i|meas)} = \frac{P(meas|occ_{i-1})}{P(meas|free_{i-1})} \cdot \frac{P(occ_{i-1})}{P(free_{i-1})} \tag{4.4}$$

In many applications the assumption of independent measurements is wrong, but we solved this problem by only accepting measurements that overcame a threshold regarding angular distance or depth difference.
We define the likelihood quotient $lhq := \frac{P(meas|occ_{i-1})}{P(meas|free_{i-1})}$ and the result is:

$$odd_i = odd_{i-1} \cdot lhq \tag{4.5}$$

Odds relate two events to each other and thus, measure how probable one is in relation to the other avoiding scaling factors. The final modification is applying the logarithm to equation 4.5 in order to get the so-called log-odds representation:

$$\log(o_i) = \log(o_{i-1} \cdot lhq) = \log(o_{i-1}) + \log(lhq) \tag{4.6}$$

The main difference to Octomap is that specularity and distance dependent noise are considered in the update step. For more detailed information on the probability computation please see [Sup08].

4.3 Optimization

The area of optimization is concerned with the problem of finding the optimal parameters of a - in many cases - complex system. Problems and thus cost functions can either be linear or - as in our case - non-linear. The optimum is defined by the minimum or maximum of a cost function. In most cases it is not possible to find the solution analytically and therefore a numerical method is needed. There are many toolboxes that have the most common optimization algorithms already implemented, such as the Ceres Solver by Google [AMO], NLopt by the MIT (Massachusetts Institute of Technology) [Joh] or the Eigen library [GJ+]. Ceres is based on the Eigen library but due to sophisticated changes and additions it is supposed to perform a lot faster. All those toolboxes have their advantages and disadvantages, for example NLopt has a limited number of implemented algorithms and the Levenberg-Marquardt is not one of them. On the other hand Eigen has no implementation of the Nelder-Mead Simplex algorithm. We decided to try the Ceres Solver with the Levenberg-Marquardt algorithm but the disadvantage compared to the Eigen library is, that the Ceres Solver is not a header-only library and because of that we had problems getting it running on different systems. The toolbox is

dependent on installing libraries such as Google log and Google flags, which gave us compatibility issues.

We considered, studied and tested all of the previously mentioned toolboxes but in the end chose the Eigen library with its Levenberg-Marquardt implementation, as the toolbox is easily installed, well documented and widely used, and the algorithm is one of the most promising solutions.

4.3.1 Optimization Algorithms

We considered two of the most popular optimization algorithms, which are quite different - the Levenberg-Marquardt algorithm and the Nelder-Mead Simplex algorithm.

Marquardt-Levenberg optimization: is based on numerical derivatives of the cost function, which makes it fast in many cases. The performance of its final convergence, i.e. near the global minimum, is also high but not the highest. It is also good with complicated cost spaces. However, it is susceptible to being trapped by local minima.

Nelder-Mead simplex optimization: is based on geometrically generating a set of down-hill points in the cost space. It starts by randomly seeding out points around the starting values. In most cases it is more stable but not as fast as Levenberg-Marquardt. The main disadvantage of the Nelder-Mead Simplex algorithm is the fact that it sometimes gets stuck and goes in circles around a minimum. An advantage, on the other hand, is that it is more robust regarding local minima.

Levenberg-Marquardt

In this section the widely used algorithm by Levenberg and Marquardt will be explained based on [YMS03].

The Levenberg-Marquardt algorithm is a numeric optimization algorithm, which was first published by Kenneth Levenberg in 1944 [Lev44] and improved or expanded by Donald Marquardt in 1963 [Mar63]. The algorithm is a hybrid between a **gradient descent method** and the **Newton method**, which makes it a damped version based on the Newton-method that uses the principle of the least-squares method itself. These kinds of algorithms are supposed to find minima of nonlinear least-squares problems such as the following:

A set of m data points $(a_1, b_1), (a_2, b_2), \ldots, (a_m, b_m)$ is given. There is some kind of unknown,nonlinear correlation between a_i and b_i, which can be formulated as $b_i \approx f(a_i, \boldsymbol{x})$, that in addition to the variable a also depends on a parameter vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, consisting of n components ($m \geq n$). The goal in this scenario is to find the set of parameters $\boldsymbol{x}$ that minimizes the sum of errors S:

$$S = \sum_{i=1}^{m} (b_i - f(a_i, \boldsymbol{x}))^2$$

$$\Rightarrow \min_x \sum_{i=1}^{m} r_i^2, \qquad \text{with } r_i = b_i - f(a_i, \boldsymbol{x})$$

The optimality condition is given by

$$\sum_i r_i \frac{\partial r_i}{\partial x_j} = 0, \qquad \forall j \epsilon \{1, \ldots, n\}$$

Gradient descent method Gradient descent - also called steepest descent - is a first order optimization method. The goal is to compute a local minimum of a cost-function E by iteratively stepping in the direction in which the cost decreases most (fig. 4.4).

To minimize the real-valued cost $E(x) : \mathbb{R}^n \to \mathbb{R}$, the gradient for $E(x)$ is defined by the differential equation:

$$\begin{cases} x(0) = x_0 \\ \frac{dx}{dt} = -\frac{dE}{dx}(x) \end{cases}$$

Discretization:

$$x_{k+1} = x_k + \epsilon \frac{dE}{dx}(x_k), \qquad k = 1, 2, \ldots$$

In many cases the gradient descent method converges to a local minimum. For the case of a convex cost function E it will even converge to the global minimum. The step size ϵ can be changed before each iteration. The gradient descent method can be applied in a great variety of cases, but usually it is not the fastest solution.

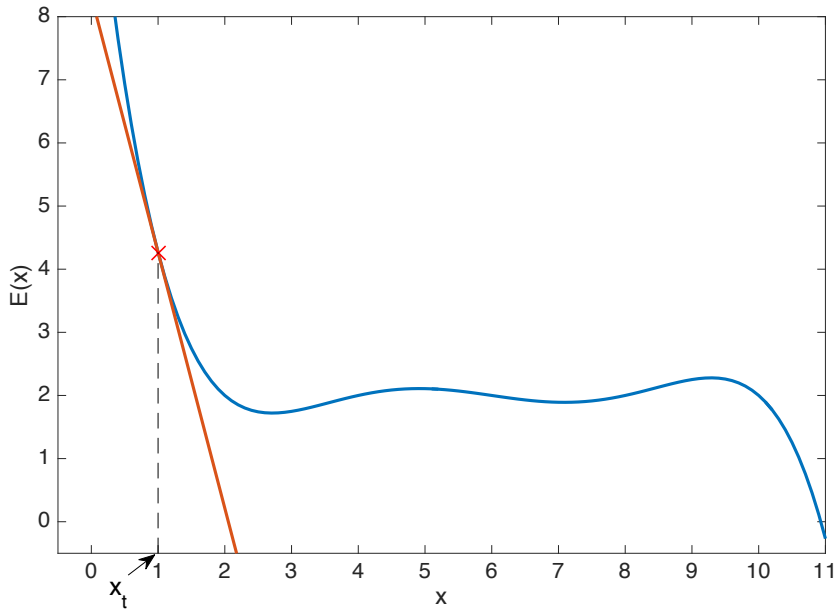


Figure 4.4: Example for the procedure of the gradient descent.

Newton method The Newton method - in contrast to the gradient descent method - is a second order optimization method and therefore takes advantage of the second derivative of the cost function. Geometrically the cost function is approximated with a quadratic function and a step towards the minimizer of that function is taken (fig. 4.5).
Using the Taylor expansion, where the first and second derivative are denoted by the Jacobian $g = \frac{dE}{dx(x_t)}$ and the Hessian $\frac{d^2E}{d^2x(x_t)}$, we get the following approximation:

$$E(x) \approx E(x_t) + g^\intercal(x - x_t) + \frac{1}{2}(x - x_t)^\intercal \boldsymbol{H}(x - x_t)$$

The corresponding optimality condition is:

$$\frac{dE}{dx} = g + \boldsymbol{H}(x - x_t) = 0$$

From this the iterative equation can be derived:

$$x_{t+1} = x_t - \boldsymbol{H}^{-1}g, \qquad t = 0, 1, \ldots \tag{4.7}$$

Second order methods tend to need less iterations than first order methods, but this does not necessarily mean that they are faster in general, because the time that is needed to compute one iteration can vary significantly. One of the computationally most expensive issues is the calculation of the inverse Hessian. A common solution for this problem is to approximate this matrix, which is the goal of the so-called quasi-Newton methods.

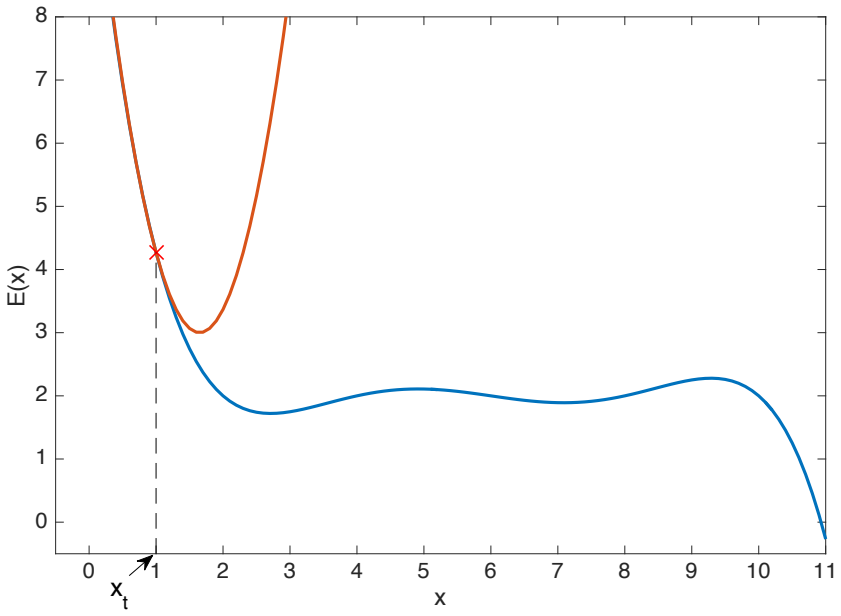


Figure 4.5: Example for the procedure of the Newton method.

The combination: Levenberg-Marquardt As mentioned previously, the Levenberg-Marquardt algorithm is a hybrid combining the gradient descent with the Newton method. In the Gauss-Newton algorithm the update step of the Newton method (eq. 4.7) $x_{t+1} = x_t - H^{-1}g$ is extended with the gradient descent

$$x_{t+1} = x_t - (H - \lambda I_n)^{-1} g$$

,which serves as the basis for the Levenberg-Marquardt algorithm. This update step realizes the combination of both methods. Without the Hessian H it would equal a gradient descent method (with step size $1/\lambda$ for $\lambda \to \infty$) and without the λI_n ($\lambda = 0$) it would equal the Newton method. So by adjusting λ, the algorithm switches smoothly between both methods.
Levenberg introduced a damped version of the Gauss-Newton algorithm in 1944:

$$x_{t+1} = x_t + \Delta, \qquad \text{with } \Delta = -(\boldsymbol{J}^\intercal \boldsymbol{J} + \lambda \boldsymbol{I}_n)^{-1} \boldsymbol{J}^\intercal r$$

Almost 20 years later - in 1963 - Marquardt proposed a more adaptive component-wise damping of this form:

$$\Delta = -(\boldsymbol{J}^\intercal \boldsymbol{J} + \lambda \; diag(\boldsymbol{J}^\intercal \boldsymbol{J}))^{-1} \boldsymbol{J}^\intercal r$$

He argued that his changes lead to a faster convergence in practice, because in many cases the Jacobian becomes relatively small and thus the gradient descent dominates with Marquardt's method. By replacing the identity matrix with the diagonal of the Jacobian product, the gradient descent is no longer favored over the Newton method.

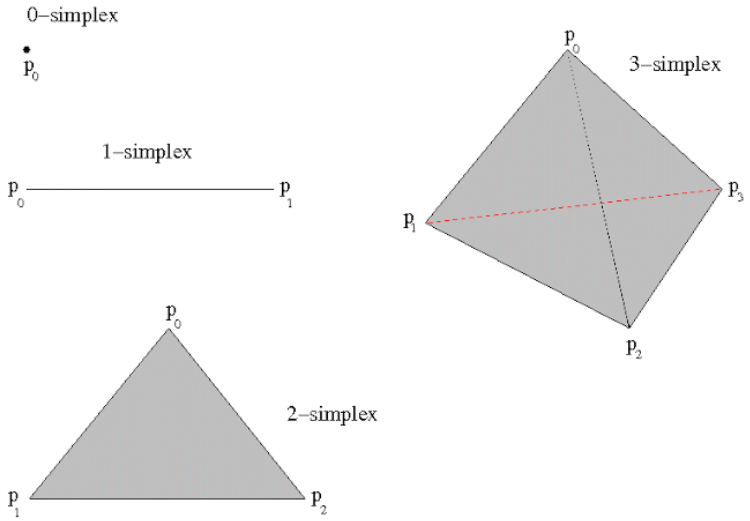


Figure 4.6: The simplex in 0, 1, 2 and 3 dimensions [Swe].

Nelder-Mead Simplex

The Nelder-Mead algorithm [NM65] is a derivative-free method of non-linear optimization, which was presented by John Nelder and Roger Mead in 1965. In n-dimensional space it uses the simplest polytopes, which can be combined using $n+1$ vertices (fig. 4.6). Each of the n dimensions represents one parameter that can be adapted in order to minimize a cost function $f(x_i)$, with $i = 0, \ldots, n$.

n = 0 (0 dimensions) the simplex is represented by a point (1 vertex)
n = 1 (1 dimension) the simplex is represented by a line (2 vertices)
n = 2 (2 dimensions) the simplex is represented by a triangle (3 vertices)
n = 3 (3 dimensions) the simplex is represented by a tetrahedron (4 vertices)
$\vdots$

In the following the procedure of the algorithm will be explained based on [Alt02]. The first step the algorithm starts with a given simplex S_0; In each iteration we:

- evaluate the cost function for each vertex $(x_1, \ldots, x_n)$ of the current simplex S_k in order to find the most expensive (worst) solution $f(x_m) = max\{f(x_0), \ldots, f(x_n)\}$
- calculate a vertex with a lower function value, replace the vertex x_m with the new vertex and thus get a new simplex S_{k+1}

In the original presentation [NM65] 3 different options for calculating the better vertex were proposed. Assume $S \subset \mathbb{R}^n$ a simplex with vertices $x_0, \ldots, x_n$ and for $j \in \{O, \ldots, n\}$

$$s_j = \frac{1}{n} \sum_{\substack{i=0 \\ i \neq j}}^{n} x_i$$

is the "center of mass" of the vertices regarding x_j (assuming the material is of uniform density).

1. **Reflection:** The vertex x_j is reflected on the "center of mass" s_j. For that we calculate the reflected point

$$x_r = s_j + \gamma \cdot (s_j - x_j)$$

with reflection constant $0 < \gamma < 1$ (fig. 4.7).

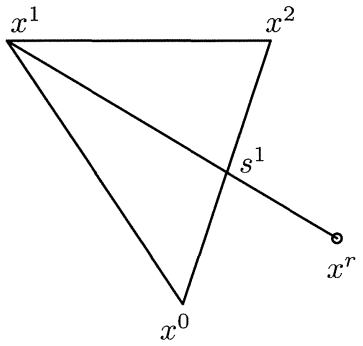


Figure 4.7: Example for the resulting parameter set produced by the Nelder-Mead Simplex algorithm by using the reflection method with $j = 1$ and $\gamma = \frac{1}{2}$ [Alt02].

2. **Expansion:** For that the vertex x_r is shifted in the direction of $s_j - x_j$ (= direction of $x_r - s_j$) resulting in a new vertex

$$x_e = s_j + \beta \cdot (x_r - s_j)$$

with expansion constant $\beta > 1$ (fig. 4.8).

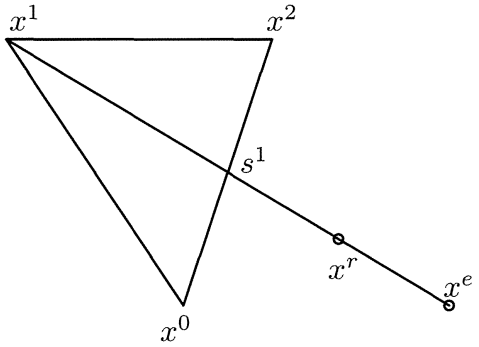


Figure 4.8: Example for the resulting parameter set produced by the Nelder-Mead Simplex algorithm by using the expansion method with $j = 1$ and $\beta = 2$ [Alt02].

3. **Contraction:** 3 types of contraction have to be distinguished:

   - Partial contraction within x_j in the direction of $s_j - x_j$ (= direction of $x_r - s_j$). The new vertex is

     $$x_c = s_j + \alpha \cdot (x_j - s_j)$$

     with the contraction constant $0 < \alpha < 1$ (fig. 4.9.

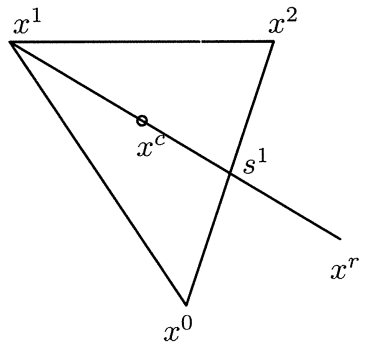


Figure 4.9: Example for the resulting parameter set produced by the Nelder-Mead Simplex algorithm by using the contraction method with $j = 1$ and $\alpha = \frac{1}{2}$ [Alt02].

   - Partial contraction outside of x_r in direction $s_j - x_r$ (= direction of $x_j - s_j$). The new vertex is

     $$x_c = s_j + \alpha \cdot (x_r - s_j)$$

     with the contraction constant $0 < \alpha < 1$ (fig. 4.10.

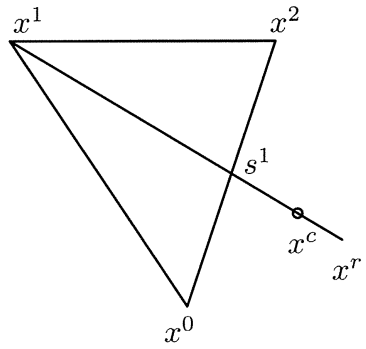


Figure 4.10: Example for the resulting parameter set produced by the Nelder-Mead Simplex algorithm by using the contraction method with $j = 1$ and $\alpha = \frac{1}{2}$ [Alt02].

- Total contraction regarding x_j: In this case all vertices x_i, $i = 0, \ldots, n$, $i \neq j$ are replaced by $\hat{x}_i$ with

$$\hat{x_i} = x_i + \frac{1}{2} \cdot (x_j - x_i) = \frac{1}{2} \cdot (x_i + x_j)$$

as depicted in fig. 4.11.

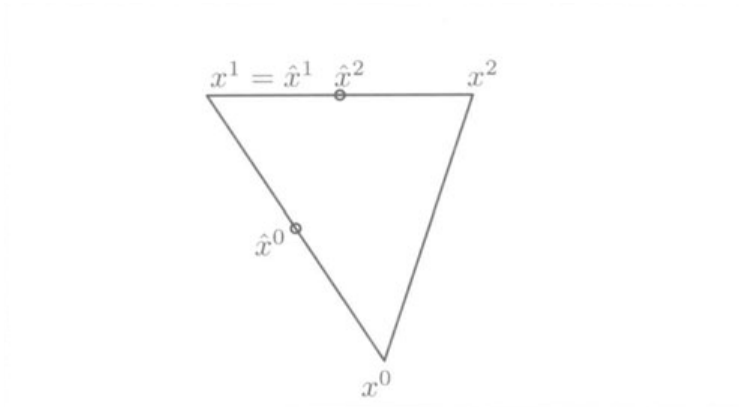


Figure 4.11: Example for the resulting parameter set produced by the Nelder-Mead Simplex algorithm by using the total contraction method with $j = 1$ [Alt02].

Now we can describe the procedure of the algorithm using these three basic construction methods.

Procedure of the Nelder-Mead Algorithm

1. Choose an initial guess $x_{(0,0)} \in \mathbb{R}^n$ and calculate the vertices of the initial simplex S_0:

$$x_{(0,j)} = x_{(0,0)} + e_j, \qquad j = 1, \ldots, n$$

where e_j are the unit vectors. Set $k := 0$.

2. Find the vertex $x_{(k,m)}$ with

$$f(x_{(k,m)} = max\{f(x_{(k,0)}, \ldots, f(x_{(k,n)}\},$$

the vertex of S_k with the greatest function value, the vertex $x_{(k,l)}$ with

$$f(x_{(k,l)} = min\{f(x_{(k,0)}, \ldots, f(x_{(k,n)}\},$$

the vertex of S_k with the smallest function value, and

$$s_{(k,m)} = \frac{1}{n} \sum_{\substack{i=0 \\ i \neq m}}^{n} x_{(k,i)},$$

the "center of mass" regarding $x_{(k,m)}$.

3. Use the reflection method to reflect $x_{(k,m)}$ on the "center of mass" $s_{(k,m)}$ and get
$$x_r = s_{(k,m)} + \gamma \cdot (s_{(k,m)} - x_{(k,m)}).$$
We expect $f(x_r) < f(x_{(k,m)})$ to be true.

4. We distinguish 3 cases:

   - $f(x_r) < f(x_{(k,l)})$, which means the reflection resulted in a new minimal vertex. In that case we try to find an even better vertex through expansion of x_r in the direction of $x_r - x_{(k,m)}$:
   $$x_e = s_{(k,m)} + \beta \cdot (x_r - s_{(k,m)})$$
   and replace $x_{(k,m)}$ with the better of both vertices x_r and x_e, which means we set
   $$x_{(k+1,m)} := \begin{cases} x_e, \text{ if } f(x_e) < f(x_r), \\ x_r, \text{ if } f(x_r) \leq f(x_e). \end{cases}$$
   All the other vertices of S_k remain the same.
   - $f(x_{(k,l)} \leq f(x_r) \leq max\{f(x_{(k,j)}|j \neq m\}$: In this case x_r is not better than all other vertices except $x_{(k,l)}$, usually better than $x_{(k,m)}$ and we replace $x_{(k,m)}$ with x_r. All other vertices of S_k remain the same.
   - $f(x_r) > max\{f(x_{(k,j)})|j \neq m\}$:
   If $f(x_r) \geq f(x_{(k,m)})$, the shift of $s_{(k,m)}$ in the direction of x_r was probably wrong and we try the opposite direction. A partial contraction of $x_{(k,m)}$ in the direction of $s_{(k,m)} - x_{(k,m)}$ is performed and we get
   $$x_c = s_{(k,m)} + \alpha \cdot (x_{(k,m)} - s_{(k,m)}).$$
   If $f(x_r) < f(x_{(k,m)})$, the shift was probably right after all but because all vertices except for $x_{(k,m)}$ are better than x_r we should try to get back closer to the simplex. Thus, we execute a contraction of x_r in the direction of $s_{(k,m)} - x_r$
   $$x_c = s_{(k,m)} + \alpha \cdot (x_r - s_{(k,m)}).$$
   If $f(x_c) < f(x_{(k,m)}$ we set $x_{(k+1,m)} := x_c$ and all other vertices of S_k remain the same. In any other case, all trials did not help and therefore a total contraction regarding $x_{(k,l)}$ is executed. For all $i \neq l$ we set
   $$x_{(k+1,i)} = \frac{1}{2} \cdot (x_{(k,i)} + x_{(k,l)}).$$
   That way the vertex with the smallest function value remains the same.

5. Set $k := k + 1$ and go back to step 2.

The popularity of this method can be explained with the fact that no derivative information is required and thus all steps are computed by simply evaluating the function. This means that for cost functions whose function evaluations are computationally significantly cheaper than calculating their derivatives, the Nelder-Mead Simplex algorithm is a well performing approach. However, it has greedy characteristics, which means that it takes the optimal solution in every single iteration without a global perspective or considering future subproblems. Therefore, the Nelder-Mead method can not guarantee to find the global minimum and can often get stuck in local minima.

Conclusion

We decided to use the Levenberg-Marquardt algorithm first, because we have more experience with it. In previous projects we found it to perform best with more-dimensional cost functions. The more dimensions a cost function has, the fewer iterations the algorithm needs to converge. The Nelder-Mead Simplex algorithm on the other hand works fine with one-dimensional cost functions, but it takes significantly more iterations to converge. The fact that one algorithm takes less iterations to converge than the other does not necessarily mean that it is also faster, because we do not know anything about the time one iteration takes respectively. The Nelder-Mead Simplex is an alternative approach, which should be considered in the future.

4.3.2 Representation of Rotation (Optimizer Parameters)

The result we expect our algorithm to deliver is a certain correction movement of our robot, which optimizes its stance with respect to the workstation. This movement has to be represented mathematically and is therefore divided into a rotation and a translation. Translations within the 3D space are typically represented by a 3D vector such as $t = \left(t_x, t_y, t_z\right)^\intercal$, rotations on the other hand can have multiple representations. One of the most popular embodiments is the rotation matrix, because it can easily be combined with the translation vector to a affine transformation matrix in homogeneous coordinates of the form:

$$\begin{pmatrix} \boldsymbol{R} & t \\ 0 & 1 \end{pmatrix}, \qquad \text{with } R \in \mathbb{R}^{3x3}$$

Rotation matrices have special properties, for example they have to be orthogonal ($R^{-1} = R^\intercal$) and their determinant has to equal one ($det(R) = \pm 1$). The main advantage of affine transformation matrices can be shown with a simple example:
Assume we want to perform a rotation around the z-axis by 30° and a translation in z-direction by $0.5m$ subsequently on the point $p_1 = (3, 5, 1)^\intercal$ within the $\mathbb{R}^3$ space. The rotation matrix that rotates a vector $p \in \mathbb{R}^3$ around the z-axis by the angle θ

is:

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

The result p_2 of the rotation would be:

$$p_2 = R_z(30°) \cdot p_1 = \begin{bmatrix} 0.866 & -0.5 & 0 \\ 0.5 & 0.866 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 5 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.098 \\ 5.83 \\ 1 \end{bmatrix}$$

The additional translation $t = (0, 0, 0.5)^\intercal$ could be realized with:

$$p_3 = p_2 + t = \begin{bmatrix} 0.098 \\ 5.83 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 0.098 \\ 5.83 \\ 1.5 \end{bmatrix}$$

So the whole transformation would be:

$$p_3 = R_z(30°) \cdot p_1 + t = \begin{bmatrix} 0.098 \\ 5.83 \\ 1.5 \end{bmatrix}$$

The combination of matrix-vector multiplication and vector-vector addition can be simplified with homogeneous coordinates. We construct a 4×4 matrix that consists of $\boldsymbol{R}$ and t and thus can compute p_3 with a simple vector-matrix product:

$$p_{3,hom} = \begin{bmatrix} \boldsymbol{R} & t \\ 0 & 1 \end{bmatrix} \cdot p_{1,hom} = \begin{bmatrix} 0.866 & -0.5 & 0 & 0 \\ 0.5 & 0.866 & 0 & 0 \\ 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 5 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.098 \\ 5.83 \\ 1.5 \\ 1 \end{bmatrix}$$

The upside is that homogeneous coordinates are easy to handle mathematically, because the procedure of multiplying a vector with the rotation matrix and adding the translation afterwards one can just multiply the extended vector with the affine transformation matrix.
The problem we had with this representation is that it is not easy to handle it for optimization. An optimizing algorithm tries to adjust a set of parameters in such a way that a cost function is minimized. The more parameters the optimizer can adjust, the more costly the computation becomes, but on the other hand more parameters often mean that less iterations are needed for the algorithm to converge. Because of that we wanted to represent the rotation with as few parameters as possible, while still getting a good convergence rate.
Two of the most popular options for compact representation of rotations are Euler angles and quaternions. These are presented in the following.

Euler angles

Euler angles were introduced by the mathematician Leonhard Euler. They are three independent parameters α, β, γ, that define the rotation angle regarding three different axes. The axes we chose were the z-, y- and x-axis, which is often referred to as yaw-pitch-roll (fig. 4.12).

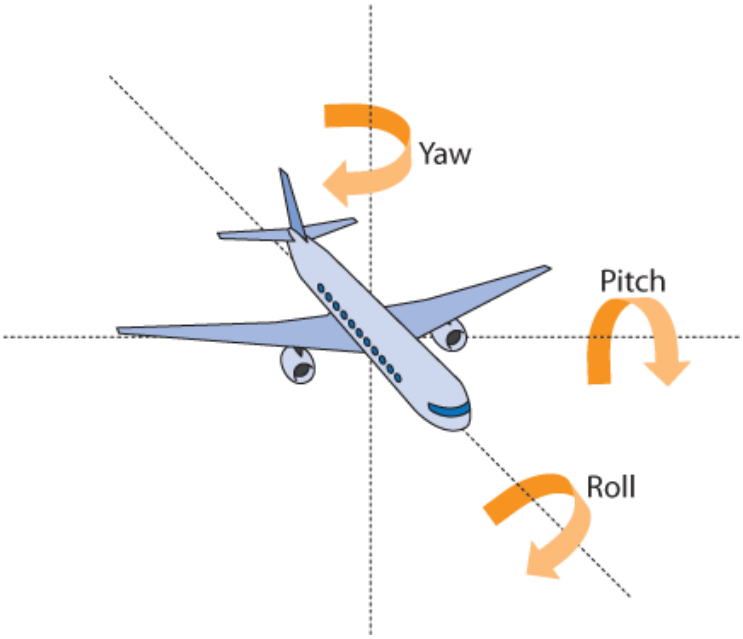


Figure 4.12: Visualization of the yaw-pitch-roll parameters [Ree10].

Euler angles can easily be converted to a rotation matrix. For that each rotation (yaw, pitch, roll) is expressed by a separate matrix $R_z(\gamma)$, $R_y(\beta)$, $R_x(\alpha)$ and the total rotation is gained by concatenation:

$$
\begin{aligned}
R_{tot} &= R_z(\gamma) \cdot R_y(\beta) \cdot R_x(\alpha) \\
&= \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 1 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{bmatrix}
\end{aligned}
$$

Euler angles are a intuitive representation of rotations, because imagining what it means to rotate an object by the exemplary Euler angles $\alpha = 90°$, $\beta = 0°$ and $\gamma = 45°$ is simple. The main disadvantage of this representation are singularities. There are multiple constellations for the parameters α, β and γ that lead to the same R_{tot} and thus to the same transformation. This phenomenon is also known as Gimbal Lock. The alternative representation - quaternions - does not have this problem.

Quaternions

Quaternions in general are nothing but a number system that extends the complex numbers and were introduced in 1843 by W. Hamilton. Their general practicality will be explained based on [YMS03].
In general the set of complex numbers $\mathbb{C}$ can be simply defined as $\mathbb{C} = \mathbb{R} + \mathbb{R}i$ with $i^2 = -1$. A similar generalization of complex numbers are quaternions. They are denoted by $\mathbb{H}$ and are defined as:

$$\mathbb{H} = \mathbb{C} + \mathbb{C} \cdot j, \text{ with } j^2 = -1 \text{ and } i \cdot j = -j \cdot i$$

From this follows that an element of $\mathbb{H}$ has the form

$$q = q_0 + q_1 i + (q_2 + q_3 i)j = q_0 + q_1 i + q_2 j + q_3 ij, q_0, q_1, q_2, q_3 \in \mathbb{R}$$

The product ij is often denoted by k due to simplicity reasons. For rotations only unit quaternions are used, so $\|q\|^2 = q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$ has to be true.
Assuming a given rotation matrix $R = e^{\hat{\omega}t}$ with $\|\omega\| = 1$ and $t \in \mathbb{R}$ the corresponding quaternion is:

$$q(R) = \cos(t/2) + \sin(t/2) \cdot (\omega_1 i + \omega_2 j + \omega_3 ij)$$

The advantage of quaternions is - as previously mentioned - that they do not have singularities. The disadvantage on the other hand is that they consist of 4 parameters and thus they would be computationally more costly than Euler angles in terms of optimization. Because of this we decided to work with a third alternative, which combines the advantages of both previous representations.

Angle-Axis representation

According to Euler's rotation theorem, any sequence of rotations about a fixed point within the 3D space can be replaced by a rotation by a certain angle θ around a fixed axis that runs through the fixed point. The axis is defined to be a unit vector $u = (u_1, u_2, u_3)^\intercal$ and thus $u_1^2 + u_2^2 + u_3^2 = 1$ has to be true. Using this representation we would still have 4 parameters just like with quaternions, but we can exploit the fact, that u is defined as a unit vector. With this knowledge we choose our representation to be the product of the rotation angle θ and the rotation axis u:

$$\theta \cdot u = \begin{bmatrix} \theta u_1 \\ \theta u_2 \\ \theta u_3 \end{bmatrix}$$

This way our optimizer can work with 3 parameters, which makes the computation faster than with 4 parameters and we still avoid the problems Gimbal Lock brings.

4.3.3 Cost Functions

The essential function of an optimization algorithm is to minimize a given cost function by changing a set of parameters. We chose a angle-axis representation for our set of parameters, because it combines the advantages of Euler angles and quaternions, while avoiding their respective disadvantages. In order to find the cost function that lead to the fastest and most robust convergence we created a simplified test scenario. A 3×3 identity matrix called *target matrix* T was rotated by various angles and axis, resulting in another matrix called *model* M. The target matrix represents our camera data and the model matrix represents the 3D model of our shelves. We gave the optimization algorithm both matrices, three different cost functions and wanted to know the transformation that has to be applied to the *model* matrix in order to minimize each cost function e_i.

1. **Frobenius norm of error matrix**
   In each iteration of the optimizing algorithm we applied the resulting transformation R to the model matrix M. The result should get closer and closer to the target matrix T ($T \approx R \cdot M$) and for the ideal transformation R, $T = R \cdot M$ is true. Thus, the first cost function of our choice was the Frobenius norm of the difference of those two matrices, which becomes zero for the optimal transformation R.

$$e_1 = \|T - R \cdot M\|_F$$

2. **Frobenius norm of difference to identity matrix**
   The second cost function of our choice was based on the knowledge that the product of a matrix A with its inverse A^{-1} equals the identity matrix I. If $T \approx R \cdot M$ is true, than we can deduce $I \approx T \cdot (R \cdot M)^{-1}$ from that. This lead us to our second cost function, for which we chose the Frobenius norm of the difference between the identity matrix and the product of our target matrix T and the inverse of the product of the resulting transformation R and our model matrix M.

$$e_1 = \|I - (T \cdot (R \cdot M)^{-1})\|_F$$

3. **9 dimensional cost function consisting of error matrix entries**
   We found that the Levenberg-Marquardt optimization algorithm requires less iterations the more dimensional the cost function is. Therefore we simplified our first cost function through leaving out the Frobenius norm and calculating the cost function element-wise from the error matrix E. E is the resulting matrix of the difference of our target matrix T and the product of the resulting transformation R and our model matrix M ($E = T - R \cdot M$).

$$
\begin{aligned}
e_1 &= E(1,1) \\
e_2 &= E(1,2) \\
&\vdots \\
e_9 &= E(3,3)
\end{aligned}
$$

All three of the above cost functions were compared in various cases. The basic idea of comparing the cost functions was based on three aspects: the numbers of iterations our optimization algorithm needed to produce a satisfying result, the final error and the limits of convergence or maximum transformation (greatest rotation angles), which still converged to a satisfying result. As an error measure we chose the angle the matrix would have to be rotated by around one certain axis, after applying the resulting transformation from the optimizer, in order to make the model matrix identical with the target matrix. As mentioned above, not only the cost function affects the performance of the optimization algorithm, but also the representation of the parameters, the algorithm has to work with (Euler angles, quaternions, angle-axis representation). Therefore, we tested every cost function with Euler angles as well as with quaternions. Finally the most promising cost function (3) was also tested with the angle-axis representation, because we wanted to be certain that the angle-axis representation performed at least as well as the quaternions and Euler angles. Another variation we tried in the case of Euler angles was to use degree values in comparison to radian values, because we assumed, that the optimizer would converge faster using degrees. The idea was that by using degrees the steps between two sets of parameters would be greater and thus, the effect of changing one parameter or the other would be "simpler to observe" for our algorithm, resulting in a quicker convergence. This assumption proved to be wrong, as no significant change could be observed (compare tables 4.3 and 4.4).
Table 4.1 shows the first and simplest test scenario, where a rotation around only one axis (x-axis) was performed. In table 4.2 the same scenario was tested using degrees instead of radian values in order to check whether the higher parameter values of the optimizer had any effect on the convergence speed. Tables 4.3 and 4.4 respectively show the results of the more complicated test. In these cases we performed a rotation around the x-, y- and z-axis at once, which made it significantly more difficult for the optimizer to find the correct solution.
The different cost functions are color coded in the tables as follows:

- Cases 1-3 used quaternions as rotation representation and cost functions 1-3.
- Cases 4-6 used Euler angles as rotation representation and cost functions 1-3.
- Case 7 used the angle-axis rotation representation and the cost function 3.

Eul.	quat.	c. fun.	it.	result				error
0.1	0.99875	1	1	1	0	0	0	0.1
0.0	0.04998	2	1	1	0	0	0	0.1
0.0	0.0	3	8	0,99875	0,04998	6.28e-26	-5.55e-26	0.0
	0.0	4	29	0.1	0.0	0.0	—	0.0
		5	29	0.1	0.0	0.0	—	0.0
		6	5	0.1	6.18e-24	-5.99e-23	—	0.0
		7	5	0.1	1.0	2.49e-25	-2.81e-25	0.0
0.2	0.99500	1	1	1	0	0	0	0.2
0.0	0.09983	2	1	1	0	0	0	0.2
0.0	0.0	3	9	0.99500	0.09983	-1.95e-24	-5.22e-26	0.0
	0.0	4	29	0.2	0.0	0.0	—	0.0
		5	29	0.2	0.0	0.0	—	0.0
		6	5	0.2	1.63e-23	3.09e-24	—	0.0
		7	5	0.2	1.0	3.83e-26	8.96e-26	0.0
0.3	0.98877	1	1	1	0	0	0	0.3
0.0	0.14944	2	1	1	0	0	0	0.3
0.0	0.0	3	11	0.98877	0.14944	2.90e-42	5.70e-44	0.0
	0.0	4	29	0.3	0.0	0.0	—	0.0
		5	29	0.3	0.0	0.0	—	0.0
		6	4	0.3	-2.87e-16	-2.87e-16	—	0.0
		7	4	0.3	1.0	-2.79e-19	-2.78e-19	0.0
0.4	0.98007	1	1	1	0	0	0	0.4
0.0	0.19867	2	1	1	0	0	0	0.4
0.0	0.0	3	11	0.98007	0.19867	1.21e-33	-1.09e-33	0.0
	0.0	4	29	0.4	0.0	0.0	—	0.0
		5	29	0.4	0.0	0.0	—	0.0
		6	6	0.4	-2.98e-34	-5.77e-34	—	0.0
		7	6	0.4	1.0	3.77e-36	2.26e-36	0.0
0.5	0.96891	1	1	1	0	0	0	0.5
0.0	0.24740	2	1	1	0	0	0	0.5
0.0	0.0	3	10	0.96891	0.24740	2.35e-25	-1.79e-24	0.0
	0.0	4	30	0.5	0.0	0.0	—	0.0
		5	30	0.5	0.0	0.0	—	0.0
		6	6	0.5	5.18e-36	-6.63e-37	—	0.0
		7	5	0.5	1.0	-3.35e-37	-1.24e-37	0.0

Table 4.1: Comparison of the performance of different cost functions with different rotation representations with increasing rotation angles (in radian) around the x-axis.

Eul.	c. fun.	it.	result				error
5	4	27	5	0	0	—	0
0	5	27	5	0	0	—	0
0	6	3	5	0	0	—	0
	7	3	5	1	0	0	0
10	4	27	10	0	0	—	0
0	5	27	10	0	0	—	0
0	6	3	10	-6.37e-19	1.86e-18	—	0
	7	3	10	1	0	0	0
15	4	27	15	0	0	—	0
0	5	27	15	0	0	—	0
0	6	4	15	0	0	—	0
	7	4	15	1	0	0	0
20	4	27	20	0	0	—	0
0	5	27	20	0	0	—	0
0	6	4	20	3.19e-26	-9.29e-26	—	0
	7	4	20	1	0	0	0
25	4	27	25	0	0	—	0
0	5	27	25	0	0	—	0
0	6	4	25	0	0	—	0
	7	4	25	1	0	0	0
30	4	27	30	0	0	—	0
0	5	27	30	0	0	—	0
0	6	6	30	3.12e-37	4.78e-37	—	0
	7	5	30	1	0	0	0
35	4	27	35	0	0	—	0
0	5	27	35	0	0	—	0
0	6	4	35	0	0	—	0
	7	4	35	1	0	0	0
40	4	27	40	0	0	—	0
0	5	27	40	0	0	—	0
0	6	5	40	6.91e-29	1.52e-29	—	0
	7	5	40	1	0	0	0
45	4	27	45	0	0	—	0
0	5	27	45	0	0	—	0
0	6	4	45	0	0	—	0
	7	4	45	1	0	0	0

Table 4.2: Comparison of the performance of different cost functions with different rotation representations with increasing rotation angles (in degrees) around the x-axis.

Eul.	quat.	c. fun.	it.	result				error
0.1	0.99638	3	16	0.99638	0.0473595	0.0523491	0.0473595	0.0
0.1	0.0473595	4	14	0.1	0.1	0.1	—	0.0
0.1	0.0523491	5	14	0.1	0.1	0.1	—	0.0
	0.0473595	6	8	0.1	0.1	0.1	—	0.0
		7	13	0.17022	0.557122	0.615817	0.557122	0.0
0.2	0.986082	3	15	0.986082	0.0889215	0.108755	0.0889215	0.0
0.2	0.0889215	4	72	0.2	0.2	0.2	—	0.0
0.2	0.108755	5	69	0.2	0.2	0.2	—	0.0
	0.0889215	6	8	0.2	0.2	0.2	—	0.0
		7	5	0.334068	0.53484	0.654135	0.53484	0.0
0.3	0.970027	3	8	0.970027	0.12402	0.168182	0.12402	0.0
0.3	0.12402	4	40	0.3	0.3	0.3	—	0.0
0.3	0.168182	5	38	0.3	0.3	0.3	—	0.0
	0.12402	6	8	0.3	0.3	0.3	—	0.0
		7	6	0.490906	0.510379	0.692118	0.510379	0.0
0.4	0.949225	3	9	0.949225	0.152145	0.229511	0.152145	0.0
0.4	0.152145	4	40	0.4	0.4	0.4	—	0.0
0.4	0.229511	5	39	0.4	0.4	0.4	—	0.0
	0.152145	6	5	0.4	0.4	0.4	—	0.0
		7	9	0.640065	0.483619	0.729538	0.483619	0.0
0.5	0.92475	3	26	0.92475	0.172955	0.291567	0.172955	0.0
0.5	0.172955	4	44	0.5	0.5	0.5	—	0.0
0.5	0.291567	5	44	0.5	0.5	0.5	—	0.0
	0.172955	6	5	0.5	0.5	0.5	—	0.0
		7	7	0.780837	0.454456	0.76612	0.454456	0.0
0.6	0.897713	3	29	0.897713	0.18628	0.353143	0.18628	0.0
0.6	0.18628	4	12	0.60902	0.59937	0.61086	—	0.00944
0.6	0.353143	5	12	0.60902	0.59937	0.61086	—	0.00944
	0.18628	6	8	0.6	0.6	0.6	—	0.0
		7	7	0.91249	0.422807	0.801542	0.422807	0.0
0.7	0.86924	3	29	0.86924	0.19213	0.413031	0.19213	0.0
0.7	0.19213	4	6	0.793625	0.682901	0.828247	—	0.10092
0.7	0.413031	5	6	0.793625	0.682901	0.828247	—	0.10092
	0.19213	6	5	0.7	0.7	0.7	—	0.0
		7	9	1.03427	0.38862	0.835434	0.38862	0.0
0.8	0.840439	3	28	0.840439	0.190689	0.47004	0.190689	0.0
0.8	0.190689	4	58	0.8	0.8	0.8	—	0.0
0.8	0.47004	5	42	0.8	0.8	0.8	—	0.0
	0.190689	6	12	0.8	0.8	0.8	—	0.0
		7	8	1.14541	0.351881	0.867383	0.351881	0.0

Table 4.3: Comparison of the performance of different cost functions with different rotation representations with increasing rotation angles (in radian) around the x-, y- and z-axis.

Eul.	c. fun.	it.	result				error
10	4	40	10	10	10	—	0
10	5	38	10	10	10	—	0
10	6	5	10	10	10	—	0
	7	5	16.7865	0.540716	0.6444	0.540716	0
20	4	47	20	20	20	—	0
20	5	38	20	20	20	—	0
20	6	7	20	20	20	—	0
	7	6	32	0.497543	0.710565	0.497543	0
30	4	38	30	30	30	—	0
30	5	42	30	30	30	—	0
30	6	8	30	30	30	—	0
	7	5	46.5675	0.447214	0.774597	0.447214	0
40	4	4	26.3443	42.0555	26.14	—	11.5651
40	5	4	26.3441	42.0555	26.1398	—	11.5652
40	6	7	40	40	40	—	0
	7	5	59.1342	0.389282	0.834817	0.389282	1.71e-06
50	4	2	43.4408	14.8979	39.8462	—	36.0694
50	5	2	43.4408	14.8979	39.8462	—	36.0694
50	6	24	50	50	50	—	0
	7	7	69.8469	0.323616	0.889126	0.323616	0
60	4	2	58.735	14.845	53.5294	—	45.4968
60	5	2	58.735	14.8451	53.5294	—	45.4967
60	6	10	60	60	60	—	0
	7	7	78.4771	0.250563	0.935113	0.250563	0

Table 4.4: Comparison of the performance of different cost functions with different rotation representations with increasing rotation angles (in degrees) around the x-, y- and z-axis.

4.3.4 Chosen Representation

The goal of our various test scenarios was to find the best performing combination of rotation representation and cost function. That meant finding a set , which converges with as little iterations as possible, while still guaranteeing a high level of robustness. Table 4.1 shows that cost functions 1 and 2 in combination with quaternions did not converge even for the smallest rotation values. We chose the quaternion $q_{init} = (1, 0, 0, 0)$ as an initial guess, because it represents a rotation of $0°$. In both cases the optimizer tended to change the non-one values in small steps ($\approx 1.5e - 18$) and stop after one iteration. Different initial guesses did not improve the outcome of the algorithm. Using the same cost functions in combination with Euler angles, the algorithm converged in many of our test cases. In more complex constellations such as a rotation around the x-, y-, and z-axis by 0.7 (rad) respectively (tab 4.3) the optimizer did not find a solution. Even though we do not expect such great transformations to occur in our application, the fact that cost function 3 needed less iterations in all test cases eliminated cost functions 1 and 2 as choices. The only decision we had left to make was to chose a rotation representation that performs well with cost function 3. As mentioned above Euler angles were no option due to the problems that might occur with the Gimbal Lock. Table 4.3 shows that the difference between 4 parameters for quaternions and 3 parameters for the angle-axis representation becomes more and more significant with increasing transformations. To find the correct solution for a rotation around the x-, y-, and z-axis by 0.8 (rad) respectively using quaternions 28 iterations are needed, using the angle-axis representation only 8 iterations are enough. In conclusion after considering the results of all test cases we chose the 9 dimensional cost function with the angle-axis rotation representation as the optimal combination.

Chapter 5

Experiments and Results

In order to quantify the performance of our algorithm, we chose to test it with another data set, which is already perfectly aligned. The new set is a model and laser scan data of a bunny statue, illustrated in figure 5.1, which can be moved arbitrarily and has a smaller file size, which makes the computations faster.

This way we were able to apply a test-wise transformation to the model, that was supposed to simulate the error between camera data and 3D model in our real-life application. Knowing the transformation that was applied to the perfectly aligned data sets gave us the possibility to check the result of our algorithm mathematically instead of just visually, because the inverse transformation was known. Despite various test cases with different transformations, initial guesses, data sets, space resolutions and parameter settings for the Levenberg-Marquardt algorithm no scenario was found that lead to convergence of the optimization algorithm. Every test case resulted in a early termination without improving the initial guess of parameters. In order to prove, that the basic idea and main contribution of our algorithm is correct and should be developed further, we chose to evaluate our cost function manually. This way it can be shown, that the problem is caused by the optimization algorithm, which can easily be exchanged. As mentioned before, the 3D model of the bunny is already perfectly aligned with the laser scans as can be seen in figure 5.2).

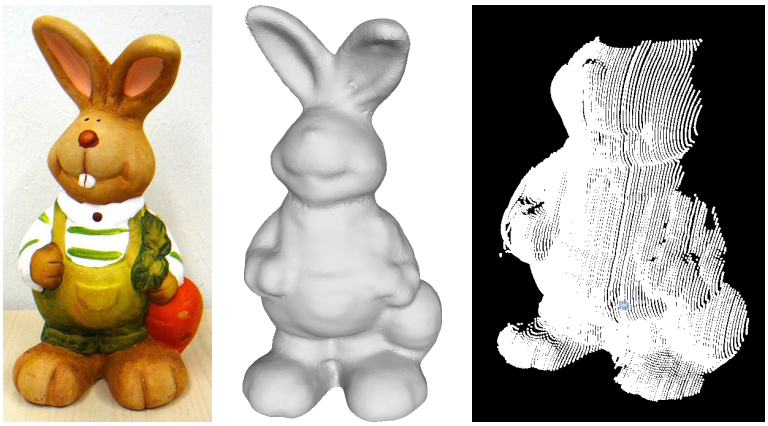


Figure 5.1: *left:* a picture of the bunny statue; *middle:* the 3D model of the bunny; *right:* an exemplary scan of the bunny.

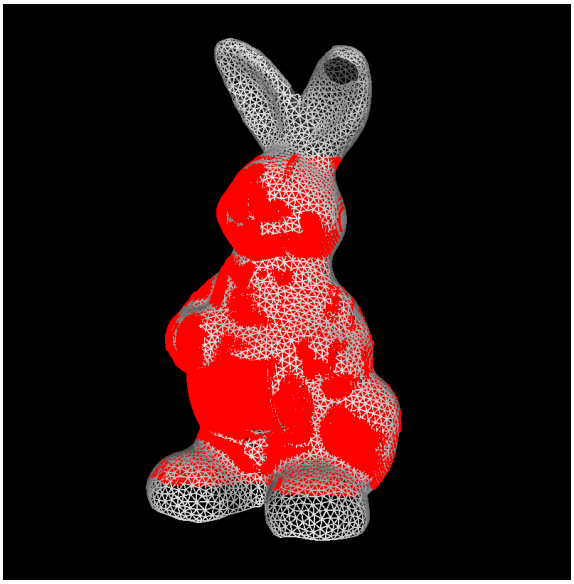


Figure 5.2: The 3D model and the scanned data are already registered to each other.

Figure 5.1 shows, that the shape of the bunny is very detailed, which means, that for a precise registration the space resolution should be high (low voxel size). The problem is that a higher resolution means a higher computational effort, which results in a trade-off between accuracy and computation speed. The bunny itself is about $20cm$ tall and about $10cm$ wide. In order to cover as many real-life scenarios as possible our test cases included rotations around x-, y- and z-axis, translations in x-, y-, and z-direction and different space resolutions.

Rotation: Figure 5.3 shows the values our cost function assumes for different rotation angles applied to our 3D model around the x-, y- and z-axis, while keeping the scanned data fixed. This equals the costs our optimization algorithm gets as a

result. It can easily be seen, that a clear global minimum can be found for a rotation of $0rad$. This means that the cost function reaches its minimum if both data sets are aligned perfectly.
In the case of rotational differences between camera data and 3D model a local optimization algorithm should be able to find the correct solution within the interval of $[-0.05, 0.1]rad$. Within this range the gradient for all curves points towards the global minimum and thus a gradient-based algorithm should converge.
Translation: Figure 5.4 shows the values our cost function assumes for different translation values applied to our 3D model in x-, y- and z-direction, while keeping the scanned data fixed. This equals the costs our optimization algorithm gets as a result for applying different translations to the 3D model. It can easily be seen, that a clear global minimum can be found for a translation of $0mm$. This means that the cost function reaches its minimum if both data sets are aligned perfectly. For the test case of translation in y-direction, a local minimum at about $-25mm$ can be observed. This can be explained easily: While moving the model forward the scan data is inside the model and less and less occupied voxels are overlapping, because the model is hollow, which leads to higher values of the cost function. As soon as the scan data gets out of the model and aligns with the backside of the bunny occupied voxels of both data sets align again. Due to the different shapes of the bunny's front and back the costs can never reach the global minimum, but still a local minimum is reached.
In the case of translational differences between camera data and 3D model a local optimization algorithm should be able to find the correct solution within the interval of $[-10, 10]mm$. Within this range the gradient for all curves points towards the global minimum and thus a gradient-based algorithm should converge. If the used algorithm is designed to avoid local minima an interval of $[-50, 30]mm$ is possible.
Space Resolution: Figure 5.5 shows the values our cost function assumes for different space resolution in the case of a rotation around the x-axis. It can be seen that with a higher resolution the costs increase as well (fig. 5.5(a)). This can be justified by the higher amount of voxels. In figure 5.5(b) both curves were plotted with differently scaled ordinate-axis to show that the general shape of the cost function is not affected by a change of space resolution.
The tables containing the values, we generated the previous diagrams with can be found in the appendix.

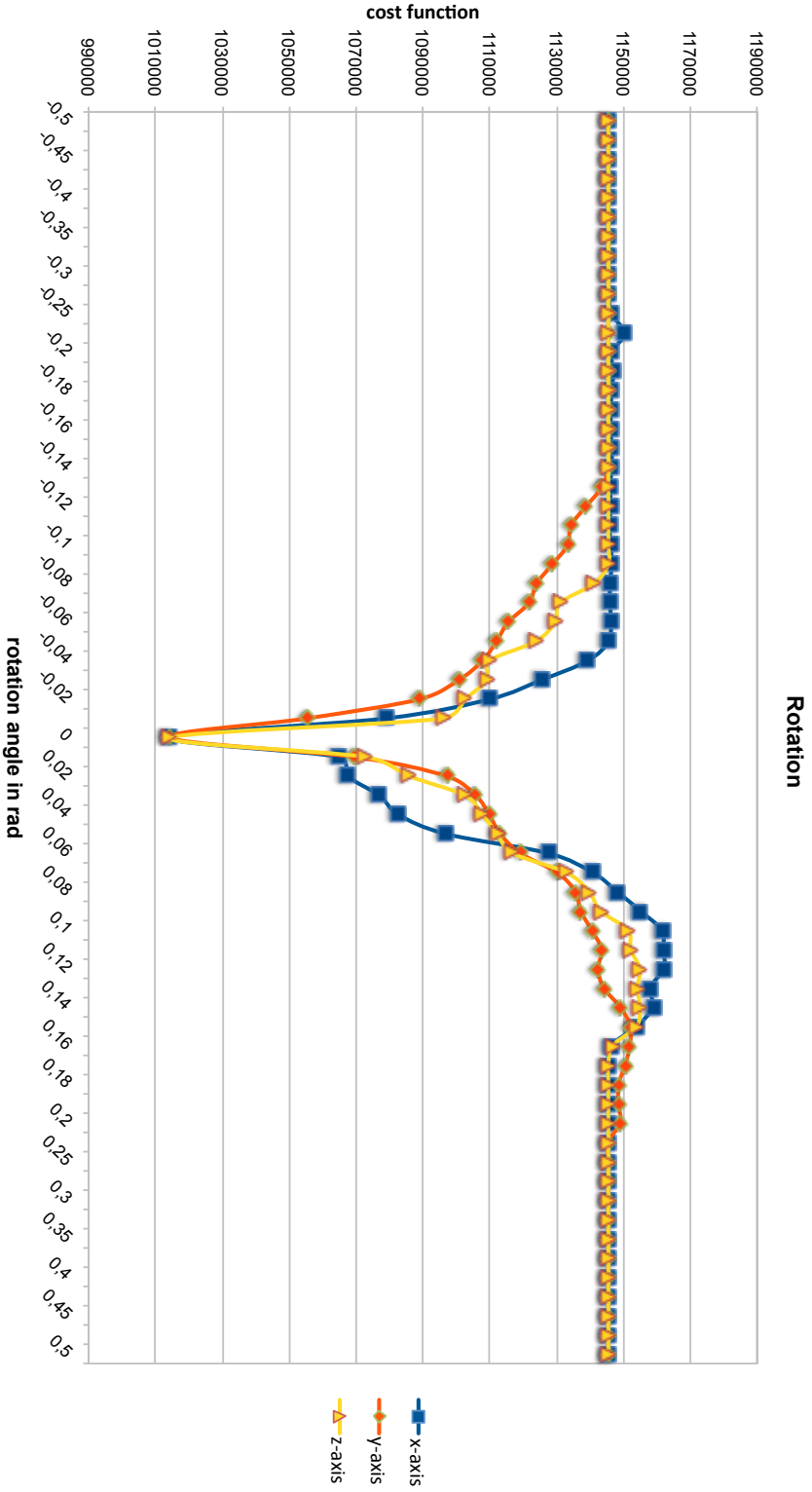


Figure 5.3: The values our cost function assumes for different applied rotations.

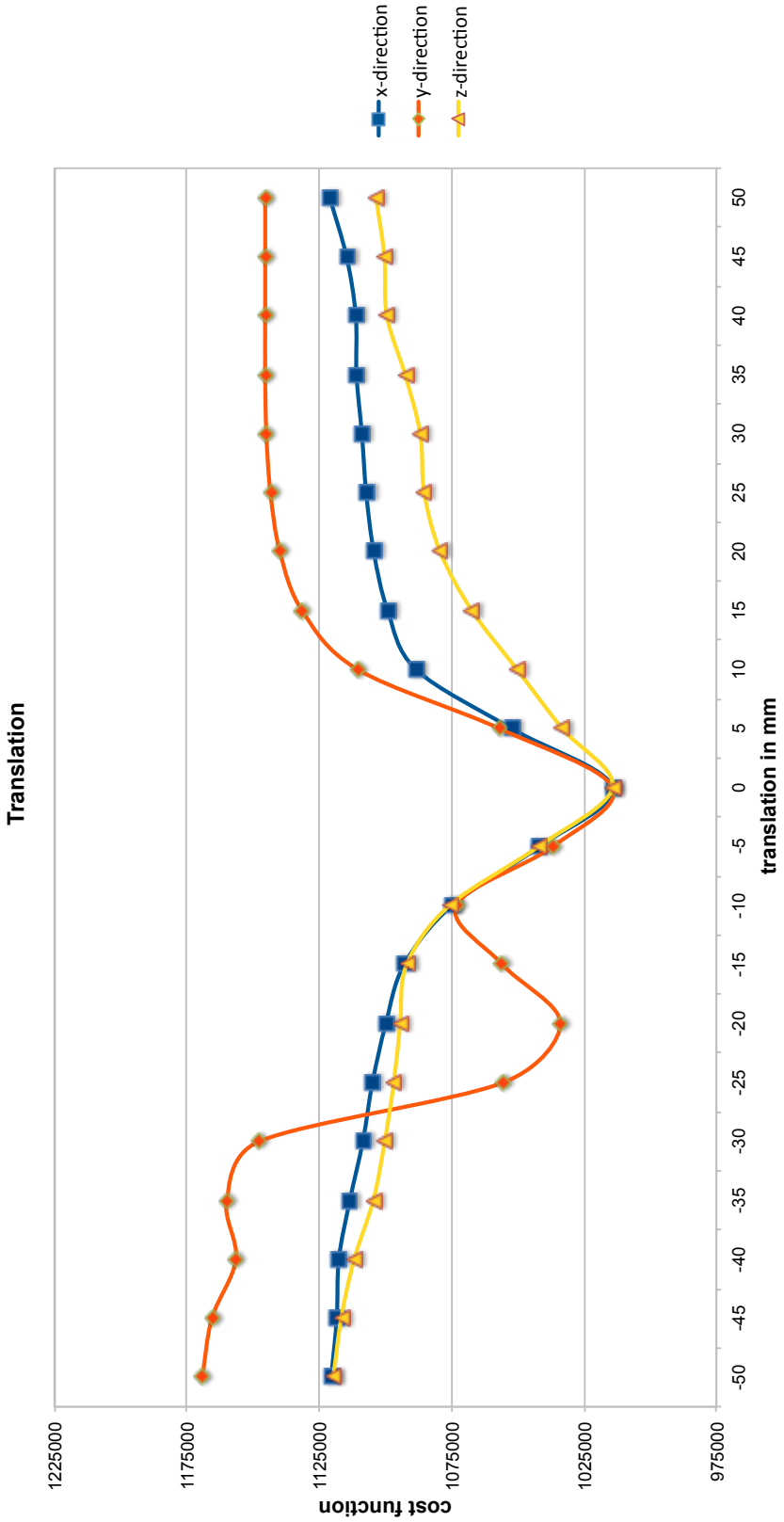
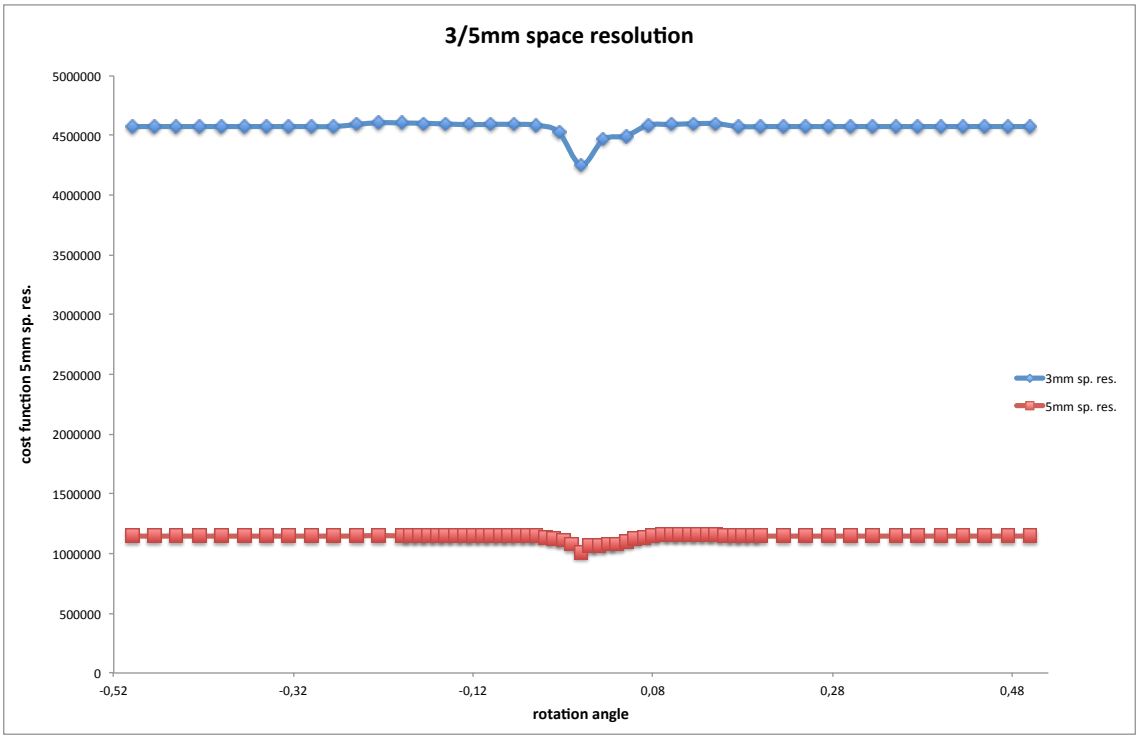
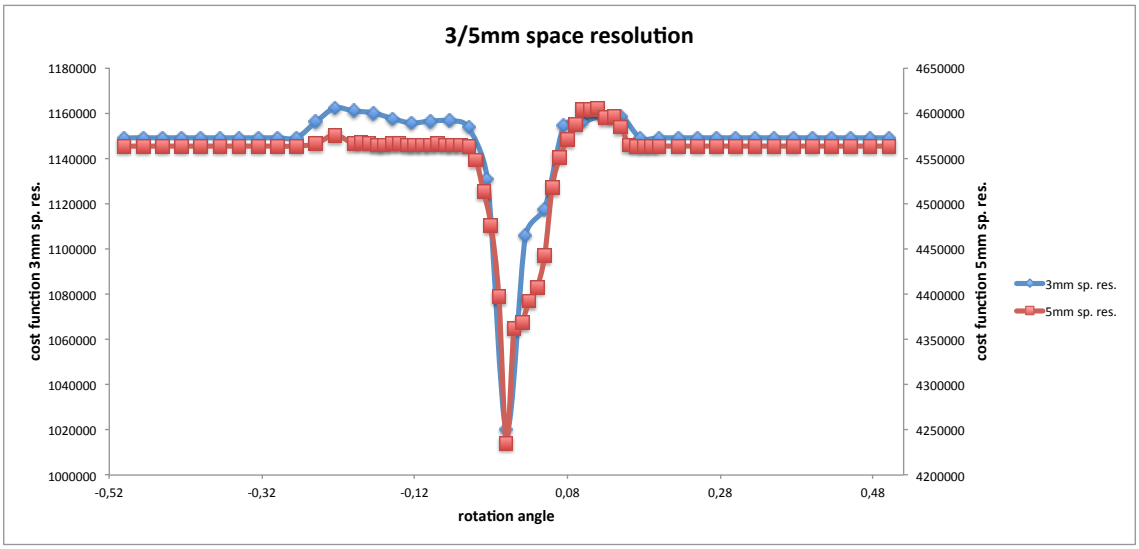


Figure 5.4: The values our cost function assumes for different applied translations.



(a) different space resolutions separately



(b) different space resolutions aligned

Figure 5.5: Visualization of the little effect a smaller space resolution has.

Chapter 6

Conclusion

The long term goal of our research group is to enable a robot to autonomously execute simple pick and place tasks. A solution using AprilTags was already found and the next step is to use a local registration approach to gain more flexibility and robustness. The idea of using a probabilistic representation for the 3D data is the main difference to existing procedures such as the ICP algorithm. In the following we will show how well our first version performed and explain what can be done to improve the performance.

6.1 Future Work/Outlook: Further Development and Comparison

In the previous section we were able to prove, that with the right optimization algorithm and settings a registration based on 3D probabilistic spaces is possible. There are several areas of improvement, that could help bringing this approach to a level at which it can compete with algorithms such as the ICP.
The most important part that needs to be improved is the optimization algorithm. The Eigen library's unsupported implementation of the Levenberg-Marquardt algorithm is cryptic, which means, that the exact meaning of exit codes or effects of parameters is not always clear. This makes debugging and finding the optimal set of parameters hard. One option would be to use a different implementation such as the previously mentioned version of Google's ceres or reading up on the source code. Another option would be to choose a totally different optimization algorithm such as the previously proposed Nelder-Mead Simplex algorithm. The best solution would be to test several optimization algorithms on different data sets and find the best match. It might be that some optimizers will find fast solutions but lack accuracy or vice versa and every application has its own priorities regarding these ares.
Ideas from registration using a truncated distance function (TSDF) could be incorporated to improve convergence even more.
Another area worth exploring is the effect of different cost functions on the real

application. We already saw in our test scenarios that the performance of different cost functions varies a lot. Some questions that should be answered are: Is it a lot faster to consider occupied voxels of one data set or the other exclusively? If yes, should the occupied voxels of the 3D model or the camera data be used? Does excluding voxels, which have the state "unknown" change the convergence speed?
As soon as a widely applicable level of development is reached the focus should be shifted towards performance and speed. In the development of the ICP the same path was explored. First a rudimentary version was published and today, after over 40 years of improvement and research we have a fast, robust and flexible algorithm for local registration.
With a working local registration algorithm optimizing within a probabilistic 3D space a comparison regarding performance, robustness and flexibility with algorithms such as the ICP and NDT could be interesting and might reveal more areas of development and improvement.

Appendix - Results

rotation angle	error x-axis	error y-axis	error z-axis
-0.5	1145367	1145367	1145367
-0.475	1145367	1145367	1145367
-0.45	1145367	1145367	1145367
-0.425	1145367	1145367	1145367
-0.4	1145367	1145367	1145367
-0.375	1145367	1145367	1145367
-0.35	1145367	1145367	1145367
-0.325	1145367	1145367	1145367
-0.3	1145367	1145367	1145367
-0.275	1145367	1145367	1145367
-0.25	1146388	1145367	1145367
-0.225	1150219	1145368	1145367
-0.2	1146399	1145369	1145367
-0.19	1146907	1145369	1145367
-0.18	1146392	1145371	1145367
-0.17	1146140	1145368	1145367
-0.16	1146138	1145373	1145368
-0.15	1146394	1145378	1145369
-0.14	1146393	1145396	1145370
-0.13	1145881	1143435	1145370
-0.12	1146138	1138652	1145372
-0.11	1145625	1134179	1145373
-0.1	1146136	1133243	1145379
-0.09	1146394	1128634	1145395
-0.08	1145883	1123962	1141114
-0.07	1145885	1121686	1130972
-0.06	1146150	1115490	1129457
-0.05	1145469	1111941	1123682
-0.04	1139298	1107808	1109884
-0.03	1125446	1100729	1109073
-0.02	1109951	1089008	1102554
-0.01	1078929	1055232	1096122
0	1014253	1014253	1014253
0.01	1064934	1069367	1072775
0.02	1067488	1097077	1085826
0.03	1076947	1105788	1102520
0.04	1082705	1109776	1107691
0.05	1096903	1113086	1112145
0.06	1127410	1119051	1116796
0.07	1140711	1130044	1132815
0.08	1148076	1135554	1139657
0.09	1154977	1137151	1143066
0.1	1161372	1140426	1151327
0.11	1161853	1143411	1152045
0.12	1161996	1142317	1154840
0.13	1157907	1144045	1154073
0.14	1158681	1149008	1154834
0.15	1153837	1152028	1153786
0.16	1146165	1151763	1146642
0.17	1145379	1150479	1145367
0.18	1145369	1148442	1145367
0.19	1145367	1148440	1145367
0.2	1145367	1148938	1145367
0.225	1145367	1145367	1145367
0.25	1145367	1145367	1145367
0.275	1145367	1145367	1145367
0.3	1145367	1145367	1145367
0.325	1145367	1145367	1145367
0.35	1145367	1145367	1145367
0.375	1145367	1145367	1145367
0.4	1145367	1145367	1145367
0.425	1145367	1145367	1145367
0.45	1145367	1145367	1145367
0.475	1145367	1145367	1145367
0.5	1145367	1145367	1145367

Space resolution 5mm		Space resolution 3mm	
rotation angle	error x-axis	rotation angle	error x-axis
-0.5	1145367	-0.5	4572606
-0.475	1145367	-0.475	4572606
-0.45	1145367	-0.45	4572606
-0.425	1145367	-0.425	4572606
-0.4	1145367	-0.4	4572606
-0.375	1145367	-0.375	4572606
-0.35	1145367	-0.35	4572606
-0.325	1145367	-0.325	4572606
-0.3	1145367	-0.3	4572606
-0.275	1145367	-0.275	4572606
-0.25	1146388	-0.25	4589736
-0.225	1150219	-0.225	4605412
-0.2	1146399	-0.2	4602316
-0.19	1146907	-0.175	4600256
-0.18	1146392	-0.15	4594120
-0.17	1146140	-0.125	4589002
-0.16	1146138	-0.1	4591039
-0.15	1146394	-0.075	4591291
-0.14	1146393	-0.05	4584180
-0.13	1145881	-0.025	4526926
-0.12	1146138	0	4250221
-0.11	1145625	0.025	4464638
-0.1	1146136	0.05	4494096
-0.09	1146394	0.075	4585208
-0.08	1145883	0.1	4590093
-0.07	1145885	0.125	4596709
-0.06	1146150	0.15	4597188
-0.05	1145469	0.175	4572607
-0.04	1139298	0.2	4572606
-0.03	1125446	0.225	4572606
-0.02	1109951	0.25	4572606
-0.0	1078929	0.275	4572606
0	1014253	0.3	4572606
0.01	1064934	0.325	4572606
0.02	1067488	0.35	4572606
0.03	1076947	0.375	4572606
0.04	1082705	0.4	4572606
0.05	1096903	0.425	4572606
0.06	1127410	0.45	4572606
0.07	1140711	0.475	4572606
0.08	1148076	0.5	4572606
0.09	1154977		
0.01	1161372		
0.11	1161853		
0.12	1161996		
0.13	1157907		
0.14	1158681		
0.15	1153837		
0.16	1146165		
0.17	1145379		
0.18	1145369		
0.19	1145367		
0.2	1145367		
0.225	1145367		
0.25	1145367		
0.275	1145367		
0.3	1145367		
0.325	1145367		
0.35	1145367		
0.375	1145367		
0.4	1145367		
0.425	1145367		
0.45	1145367		
0.475	1145367		
0.5	1145367		

translation in mm	error x-axis	error y-axis	error z-axis
-50	1120531	1168977	1119544
-45	1118347	1165519	1116677
-40	1117584	1156438	1111770
-35	1113408	1159860	1104547
-30	1108510	1147605	1100229
-25	1104858	1055365	1096808
-20	1099818	1033877	1094381
-15	1092865	1055839	1091924
-10	1074824	1073170	1075781
-5	1041795	1036759	1042185
0	1014253	1014253	1014253
5	1052385	1056962	1033208
10	1088382	1110409	1050254
15	1099145	1131434	1067325
20	1104399	1139845	1079329
25	1107246	1143399	1085313
30	1108878	1144949	1086819
35	1110899	1145391	1092540
40	1111171	1145380	1099449
45	1114658	1145372	1100353
50	1120985	1145372	1103523

Appendix - System Overview

In this chapter a short overview over the various components used in our setup will be given and the most important characteristics are listed.

miiwa

The robot we used for this work was the KUKA miiwa. The miiwa consists of the KMP omniMove 100 platform (fig.6.1) and a KMR iiwa 14 R820 lightweight robot arm (fig.6.2) mounted on top. The omniMove is equipped with two LASER scanners, which the robot uses to localize itself within a given or prerecorded 2D map, eight IR sensors, which are used for obstacle avoidance and omni-directional Mecanum wheels (fig.6.3) for locomotion. The iiwa has seven revolute joints, therefore seven degrees of freedom and can manipulate loads with a weight of up to 14 kg (payload).

Figure 6.1: The miiwa base, which serves as a mobile platform.

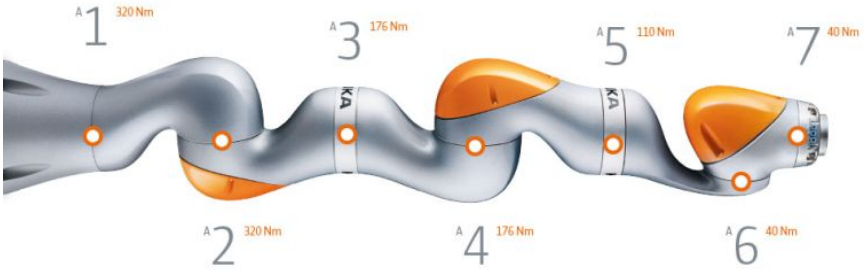


Figure 6.2: The iiwa robotic arm.

Figure 6.3: The omni-directional Mecanum wheel, also called Ilon wheel after its inventor, Bengt Ilon, who came up with the concept in 1973 working as an engineer for Mecanum AB.

Pan-Tilt

At DLR, multiple additional sensors were integrated in order to widen the variety of the robot's abilities and possibilities to perceive its environment. The iiwa was extended by a Schunk WSG-50 gripper (fig.6.4), which was mounted on the end-effector and two TCP cameras (Mako G-125 fig.6.5), which can provide 30 fps at 1.2 megapixels.
Further than that a pole was fixed to the omniMove platform with a Schunk PW pan-tilt unit (fig.6.6) on top. The pan-tilt unit carries another two cameras used for stereo vision (Manta G 201 fig.6.7), which are capable of recording at 30 fps with a resolution of 2 megapixels. Next to the cameras a Asus Xtion (fig.6.8) was placed in order to improve the perception of untextured surfaces using the IR projector inside. The whole setup can be seen in fig.1.2.

Figure 6.5: The Mako camera mounted on the end-effector.

Figure 6.4: The Schunk WSG-50 gripper mounted on the end-effector.

Figure 6.6: The Schunk pan-tilt unit mounted on a pedestal.

Figure 6.7: The Manta camera mounted on the pan-tilt unit.

Figure 6.8: The Asus Xtion mounted next to the Manta cameras on top of the pan-tilt unit.

List of Figures

Bibliography

[Alt02] Werner Alt. *Nichtlineare Optimierung: Eine EinfÃ¼hrung in Theorie, Verfahren und Anwendungen*. Springer Verlag, 2002.

[AMO] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. `http://ceres-solver.org`.

[AMT+12] Aitor Aldoma, Z-C Marton, Federico Tombari, Walter Wohlkinger, Christian Potthast, Bernhard Zeisl, Radu Bogdan Rusu, Suat Gedikli, and Markus Vincze. Tutorial: Point cloud library: Three-dimensional object recognition and 6 dof pose estimation. *Robotics & Automation Magazine, IEEE*, 19(3):80–91, 2012.

[BM92] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, 1992.

[BS03] Peter Biber and Wolfgang Straßer. The normal distributions transform: A new approach to laser scan matching. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, pages 2743–2748. IEEE, 2003.

[DKBS15] Andreas Dömel, Simon Kriegel, Manuel Brucker, and Michael Suppa. Autonomous pick and place operations in industrial production - 12th international conference on ubiquitous robots and ambient intelligence (urai). 2015.

[For] ForceFlow.be. Octree structure. `http://www.forceflow.be/2012/04/20/ray-octree-traversal/`.

[GJ+] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. `http://eigen.tuxfamily.org`.

[Heb89] Martial Hebert. Terrain modeling for autonomous underwater navigation. In *Proceedings of the 7th International Symposium on Unmanned Untethered Submersible Technology*, 1989.

[Joh] Steven G. Johnson. The nlopt nonlinear-optimization package. `http://ab-initio.mit.edu/nlopt`.

[Kri15] Simon Kriegel. *Autonomous 3D Modeling of Unknown Objects for Active Scene Exploration*. PhD thesis, München, Technische Universität München, Diss., 2015.

[Lev44] K. Levenberg. A method for the solution of certain problems in least squares. *Quaterly Journal on Applied Mathematics*, pages 164–168, 1944.

[Mar63] Donald W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, pages 431–441, 1963.

[NM65] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, pages 308–313, 1965.

[Ols10] Edwin Olson. Apriltag: A robust and flexible multi-purpose fiducial system. Technical report, University of Michigan APRIL Laboratory, 2010.

[Ree10] Aaron Reed. *Learning XNA 4.0*. O'Reilly Media, Inc., 2010.

[RL01] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling*, pages 145–152. IEEE, 2001.

[SAS+13] Jari Saarinen, Henrik Andreasson, Todor Stoyanov, Juha Ala-Luhtala, and Achim J. Lilienthal. Normal distributions transform occupancy maps: Application to large-scale online 3d mapping. *International Conference on Robotics and Automation*, 2013.

[Sup08] Michael Suppa. *Autonomous robot work cell exploration using multisensory eye-in-hand systems*. PhD thesis, Gottfried Wilhelm Leibniz Universitä t Hannover, Diss., 2008.

[Swe] Emanuel Swedenborg. The seven dimensions of physics. `http://dream-prophecy.blogspot.de/2013/02/the-seven-dimensions-of-physics.html`.

[TPB06] Rudolph Triebel, Patrick Pfaff, and Wolfram Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. In *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.

[WHB+10] Kai M. Wurm, Armin Hornung, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems. In *In Proc. of the ICRA 2010 workshop*, 2010.

[wur] wurth.at. Small load carrier. http://www.wuerth-industrie.at/web/media/pictures/wuerthindustrie/cteilemanagement/lagersichtkasten_res_wl2_530.gif.

[YMS03] Jana Kosecka Yi Ma, Stefano Soatto and Shankar S. Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models*. Springer Verlag, 2003.

License