UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

Modified Particle Swarm Optimization for a 6 DOF Local Pose Estimation Algorithm by Using a RGB-D Camera

Masterarbeit
zur Erlangung des Grades
MASTER OF SCIENCE
im Studiengang Computervisualistik

vorgelegt von

Susanne Thierfelder

Betreuer: Dipl.-Inform. Jens Hedrich, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau
Erstgutachter: Dipl.-Inform. Jens Hedrich, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau
Zweitgutachter: Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau

Koblenz, im April 2012

Kurzfassung

Particle Swarm Optimization ist ein Optimierungsverfahren, das auf der Simulation von Schwärmen basiert. In dieser Arbeit wird ein modifizierter Algorithmus, der durch Khan et al. 2010 eingeführt wurde, zur Schätzung der lokalen Kamerapose in 6 DOF verwendet. Die Poseschätzung basiert auf kontinuierlichen Farb- und Tiefendaten, die durch einen RGB-D Sensor zur Verfügung gestellt werden. Daten werden von unterschiedlichen Posen aufgenommen und als gemeinsames Model registriert. Die Genauigkeit und Berechnungsdauer der Implementierung wird mit aktuellen Algorithmen verglichen und in unterschiedlichen Konfigurationen evaluiert.

Abstract

Particle swarm optimization is an optimization technique based on simulation of the social behavior of swarms. The goal of this thesis is to solve 6 DOF local pose estimation using a modified particle swarm technique introduced by Khan et al. in 2010. Local pose estimation is achieved by using continuous depth and color data from a RGB-D sensor. Datasets are aquired from different camera poses and registered into a common model. Accuracy and computation time of the implementation is compared to state of the art algorithms and evaluated in different configurations.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Vereinbarung der Arbeitsgruppe für Studien- und Abschlussarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.	ja ☒	nein ☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	ja ☒	nein ☐

Koblenz, den 26. April 2012

Danksagung

An dieser Stelle möchte ich mich bei Prof. Paulus sowie Jens Hedrich für die intensive Unterstützung und Betreuung dieser Masterarbeit als auch für deren Vertrauen und Gewährung von Freiheiten bedanken. Ein weiterer Dank geht an die Mitarbeiter der Arbeitsgruppe Aktives Sehen und ganz besonders an Peter Decker für die anregenden Gespräche und Diskussionen. Zudem möchte ich mich für die unabhängige und unkomplizierte Unterstützung durch die Stipendienförderung der "Stiftung Industrieforschung" bedanken, die es mir ermöglicht hat mich voll und ganz auf die Erstellung dieser Arbeit zu konzentrieren.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Particle swarm optimization (PSO) is an optimization technique based on simulation of the social behavior of swarms like a flock of birds or school of fish. A swarm is a collection of particles, which is characterized by stochastic behavior synchronized with certain movement rules. These rules are derived from each particle's history and a selection of neighbors.

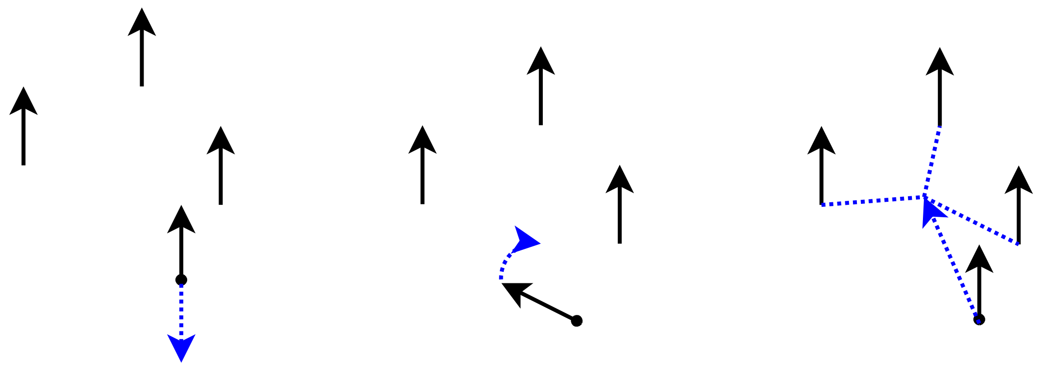


Figure 1.1: Movement behavior of particles: seperation, alignment and cohesion [Rey87]. Particles are represented as arrows. The dotted blue arrow indicates the desired movement direction of the particle, which is marked with a black dot in its origin.

Members of the swarm stay seperate by moving away from each other if they are located close to other particles. However, they head in the same general direction as other members. This is called alignment of the swarm. In the meantime particles keep the swarm together to assure cohesion (see Figure 1.1).

The reason for this behavior is considered as an evolutionary survival strategy: avoidance of collisions, protection from predators or as an effective search pattern for food. Collision avoidance is realized simply by moving away from each other.

Protection from predators is based on the confusion about their unique goal due to the apparently random movement of a huge number of preys. It makes the predators impossible to focus on a single target and predict their movement in order to adapt their own behavior. The reason for coordinated swarm behavior as a search pattern for food is the most interesting aspect of swarm simulation. It led Eberhart and Kennedy [KE95] in 1995 to the idea to introduce PSO as an optimization algorithm. The food target of natural swarms was replaced by a target optimum for the optimization algorithm. The members of the swarm are distributed in search-space and represent possible solutions.

PSO is able to optimize nonlinear functions. It has become popular, because it is simple and efficient to compute. Furthermore, it is adaptable to various problems [NdMM06]. In this thesis PSO is exploited to determine a 6 DOF pose estimation. Each particle represents a candidate pose solution of the movement of the camera in 6 DOF space. The pose of a camera is given by translation and rotation in $\mathbb{R}^3$, respectively (x, y, z, r_x, r_y, r_z) (see Figure 1.2).

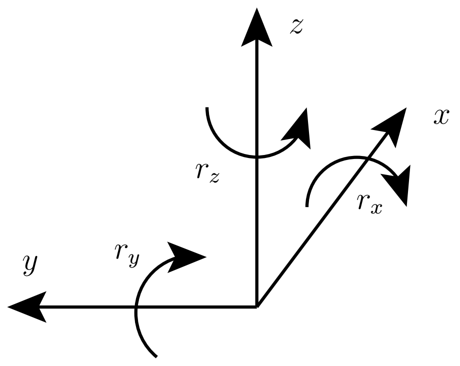


Figure 1.2: Search-space of particles in 6 DOF

The optimization algorithm in this work is used to find the best possible pose transformation of the camera compared to its last known pose. The general concept of optimization is described in [HHH98] as follows

> “Trying **variations** on an **initial concept** and using the **information gained** to **improve** the idea.“

In the work at hand, **variations** of the latest camera pose are evaluated. The **initial concept** is represented as the initial distribution of particles in search-space (see Figure 1.3). The **information gained** during the optimization procedure is represented as a cost or fitness value derived from the particles' positions in search-space. This value is used to further **improve** the result in the successive iterations by moving the particles towards a closer position to optima in search-space. The movement rules of particles in optimization are similar to the movement of swarm members in nature introduced at the beginning of this chapter.

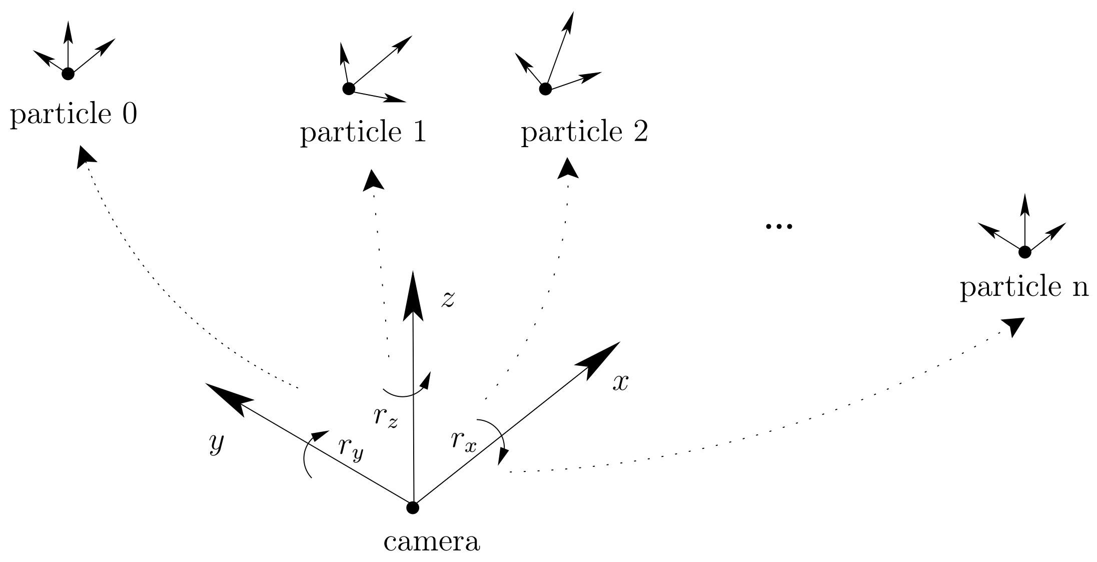


Figure 1.3: Estimation of camera movement using a particle swarm. The camera coordinate system is the last known position of the sensor with n particles as possible new pose solution of the camera

In this thesis, continuous sensor measurements containing color and depth information from a RGB-D sensor are used as input data. A similarity measurement technique has to be developed in order to compute a cost or fitness value for each particle's position in search-space. The algorithm has to be fast, robust and invariant to transformation, occlusion, and the limited field of view of this sensor. Otherwise the swarm is not able to converge with an optimal solution in an adequate amount of time.

An application for local pose estimation is 3D perception for robotics such as generation of object models or maps of the environment. The field of view of single sensor measurements without multi-view registration is too constrained in order to enable enhanced perception of the world. Therefore, datasets need to be aquired from different camera poses and registered into a common model. These models are able to serve as a basis for enhanced perception of the world. However, not only robots are able to gain advantage of algorithms for 3D registration using commercial RGB-D sensors. The generation of complete and accurate 3D models of e.g. furniture are a useful tool for designers and interior architects.

1.1 Outline

In this chapter a general introduction to optimization and the problem adressed in this thesis was given. One focus of this thesis is put in techniques and applications of PSO and the other is similarity measurement techniques for RGB-D data. A visual overview of all chapters is given in Figure 1.4.

A more detailed and formal definition of 6 DOF local pose estimation using a PSO technique is given in Chapter 2 where challenges and approaches for the solution of this problem are discussed. In Chapter 3 state of the art algorithms for 3D registration are presented and compared to the approach in this thesis. A more detailed insight into PSO and to (the used) modified version, is given in Chapter 4. This is followed by Chapter 5, which describes similarity measurement techniques required for optimization using PSO. The concept of the implementation is presented in Chapter 6, where applications, configurations and visualizations related to the theoretical concepts of previous chapters are proposed. Experiments and evaluations are presented and analyzed in Chapter 7. Finally, Chapter 8 summarizes the results of this thesis, draws conclusions and gives an outlook to future work.

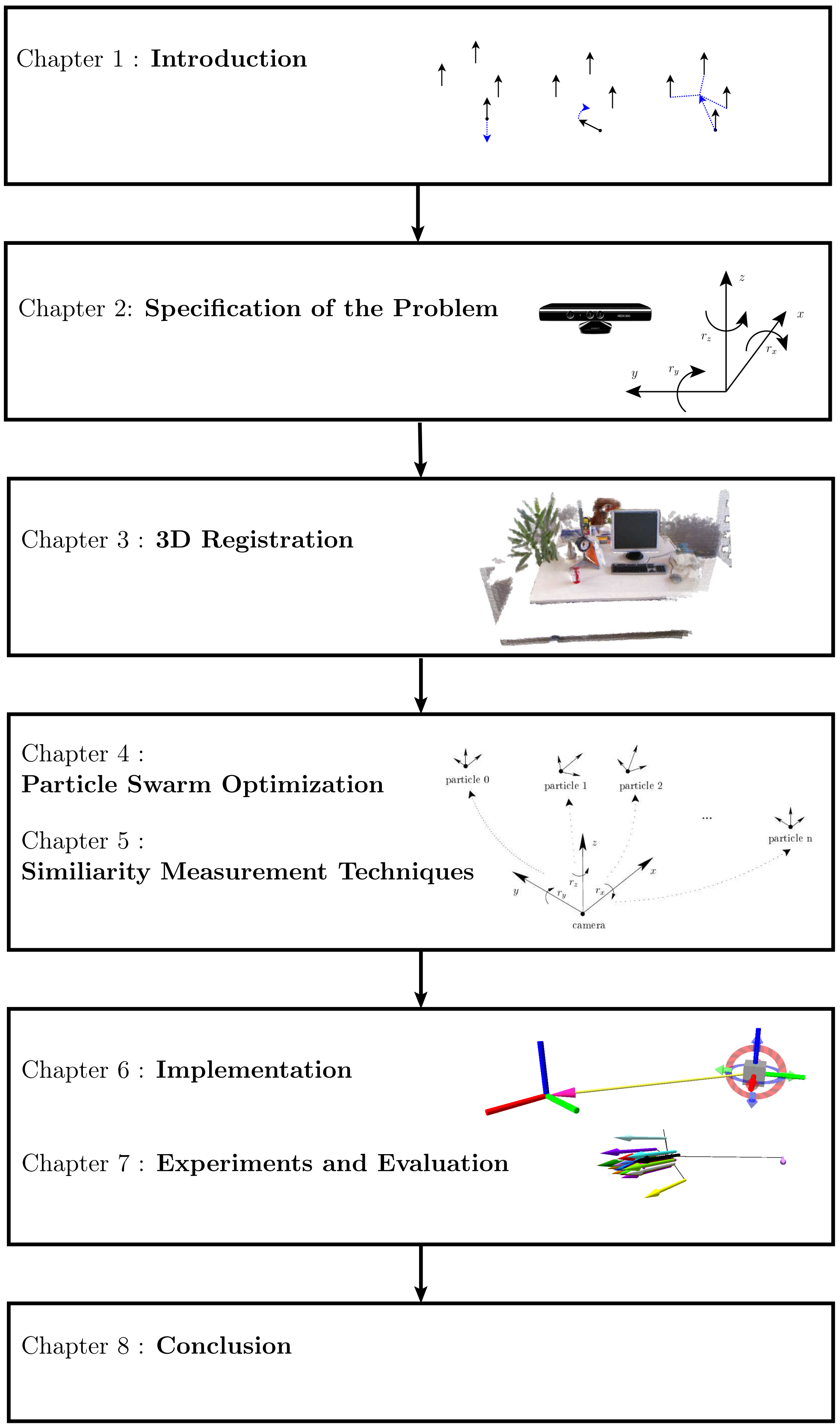

Chapter 1 : Introduction
Chapter 2: Specification of the Problem
z
x
y
r_z
r_x
r_y
Chapter 3 : 3D Registration
Chapter 4 :
Particle Swarm Optimization
Chapter 5 :
Similiarity Measurement Techniques
particle 0
particle 1
particle 2
...
particle n
camera
Chapter 6 : Implementation
Chapter 7 : Experiments and Evaluation
Chapter 8 : Conclusion

Figure 1.4: Outline of the thesis: Highly related chapters are grouped in one box.

Chapter 2

Specification of the Problems

The alignment of partial datasets from different views into a globally consistent model is called registration [Rus09]. In this work, a new dataset (floating data) is taken from a different view than the **reference data**. The **floating data** needs to be transformed to make it overlap perfectly in a common coordinate system.

2.1 Optimization

The optimization problem is stated as follows

$$T^* = \arg\min_{\alpha} O(P_1, T_\alpha(P_2)) \tag{2.1}$$

T^* computes the maximum similarity between reference P_1 and floating point cloud P_2 by altering T_α from an initial guess to the optimal solution [KN10]. The resulting transformation T^* solves the pose estimation problem. An **objective function** O evaluates the estimated solution by using a **cost or fitness function** f. The function type constitutes whether the optimization problem has to be maximized or minimized. In this thesis f is used as a cost function, which leads to f=0 as being the optimal solution. The input for registration is a stream of point clouds where each point $\boldsymbol{p}_i$ contains three-dimensional position and color information. A collection of points $\boldsymbol{p}_i$ is the point cloud structure P as in [RCG11]. All $\{x_i, y_i, z_i\}$ coordinates of $\boldsymbol{p}_i \in P$ are located in a fixed coordinate system with its origin in the center of the camera. They represent the distance from the sensor to the surface in the world. The structure of one point $\boldsymbol{p}_i$ is adaptable to the certain needs of the application, e.g. by its normal or color information. Therefore, a point in this thesis is represented as $\{x_i, y_i, z_i, r_i, g_i, b_i, n_i\}$. The data from the sensor is not assumed to be dense i.e. not every sensor measurement is valid. Furthermore, the environment is assumed to be static while the camera is allowed to move in three-dimensional space. This results in a 6 DOF local pose estimation problem.

2.2 Perception of Data

An overview of sensors for surface registration is given in [SMFF07]. Common methods for the perception of 3D point cloud data are:

- time-of-flight lasers
- laser scanning
- stereovision
- pattern projection or structured light

All techniques have in common that they never perceive full 3D information from the environment due to occlusions and the limited field of view. However, the alignment of successive sensor measurements is required for the creation of complete models from the environment. In this thesis 3D range data is provided by pattern projection from the *Microsoft Kinect* sensor [Pri] (see Figure 2.1).

Figure 2.1: Microsoft Kinect[1]

Recently, this sensor has become increasingly popular for home entertainment and gaming [HKH$^+$10]. As a consumer technology the Kinect has become affordable and therefore widely spread in the research community [IKH$^+$11].

The perception of depth information is achieved by projection of structured light into the scene as a fixed infrared pattern(see Figure 2.2). The projected IR pattern is perceived by the CMOS sensor. The depth images are computed from correspondences between projected and sensed pattern similar to stereo vision [Nüc09]. Finally, color and depth information are fused by a calibration matrix. The combination of both color and depth information allows not only to perceive an environment without any visual features but also without any illumination. Registered RGB and depth images are captured with a resolution of 640x480 at a rate of 30 frames per second[HKH$^+$10]. The accuracy of depth measurements from the Kinect is decreasing with increasing distance: it is about 1 cm at 1.5 m and 3 cm at 3 m distance [I H].The depth limit is from 0.6 m to less than 5 m and the

[1]`http://gnvr.co.uk/2011/12/03/microsoft-kinect-is-coming-to-windows/`, accessed 03.01.2012

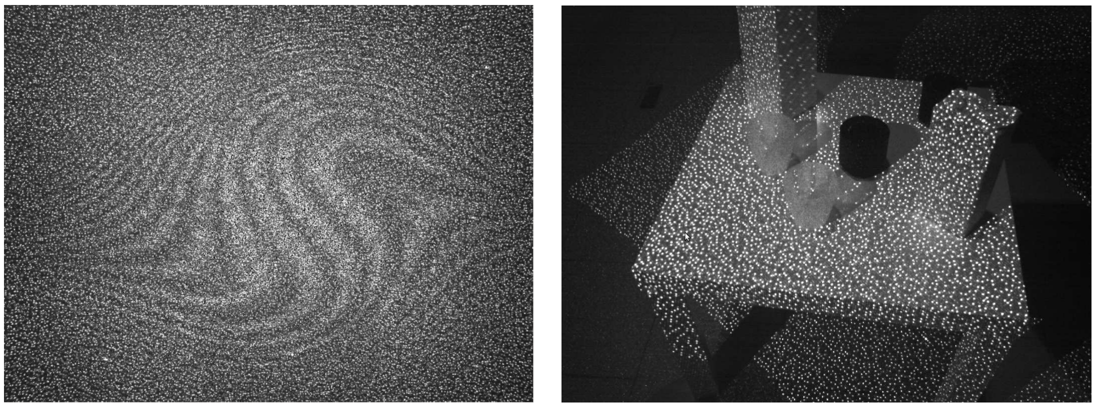


Figure 2.2: Kinect IR pattern and projection

field of view is constrained to 60 degrees. For comparision: point clouds acquired from tilting 2D or 3D laser scanners, such as the Velodyne, contain dense data with a large field of view (180 degrees) and high depth precision[MS11].

The data is often disturbed by occlusion, the limited field of view and other sensor noise like shadow borders (see Figure 2.3). The point clouds are not dense, i.e. there may be holes in the data, resulting from reflecting surfaces such as mirrors or glass, from light absorbing black surfaces or invalid distances to the measured surface [IKH+11]. The properties of the Kinect sensor is mostly comparable to a SwissRanger SR4000[2] time-of-flight laser as a 3D perception technique. However, it has a lower resolution of 176x144 pixels and is expensive. Frame rate and range are slightly higher, but also depend on the model.

2.3 Computational Challenges

In this thesis the optimization algorithm minimizes the distance between two datasets from the RGB-D sensor introduced in the previous section. However, it is difficult to find the optimum as there is no knowledge of the camera's position in the multidimensional search space (see Figure 1.2). Given two sets of points $p_i \in P_1$ and $q_j \in P_2$ one can compare the coordinates of points p_i and q_j and merge them together in a single model. However, due to sensor noise and movement of the camera, the data will almost never be identical. Missing or erroneous measurements mislead to solutions in local minima. The properties of RGB-D data challenges the algorithms required for registration. Compared to other sensors like 3D laser range finders, the data is not accurate and has a limited field of view.

[2]`http://www.mesa-imaging.ch`

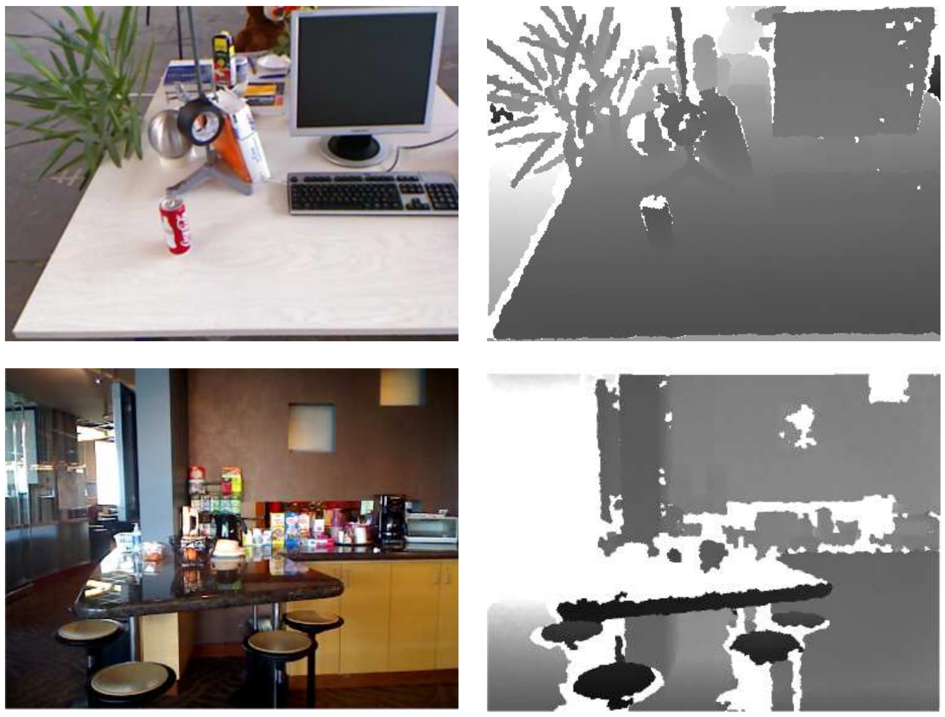


Figure 2.3: Comparison between RGB and depth image. White values in the depth image indicate that there is no valid sensor measurement. The top images are captured from the evaluation database of [SME+11] and bottom row by [HKH+10]

Though, the data rate of 30Hz enables realtime applications. However, high data rate comes with high computation time.

2.4 Approaches and Solutions

The runtime of PSO algorithms benefits from contrains in search space. In this thesis, the camera is assumed to move "naturally". This means, successive sensor measurements are assumed to be located in the vicinity to one another. Literally, the camera will not "jump" 5 meters in $\frac{1}{30}$ seconds. The more accurate the initial distribution of particles is in search space, the faster an optimum is reached. The Kinect is also equipped with an accelerometers and a tilt motor. In robotics an inertial measurement unit (imu) is used to balance sensors on moving plattforms e.g. a 2D laser range finder mounted on a pan-tilt unit for map building. The imu data can be used to enhance the initial distribution of particles in search space. If the sensor is pointing in a certain direction, the initial range of particles' positions and orientations should be set to this view. If the initial pose is set in a proper position, the optimization process will be able to converge faster and large errors in pose estimation will be avoided. It is possible to limit the amount of iterations by lowering the number of particles, which are initialized in the swarm. Due to the smart movement characteristics of particles in search space, also a few particles are able to find an optimum in 6 DOF. Futhermore, an iteration step can be parallelized. The cost value of particles benefits not only from geometric but also appearance information, which is delivered from the RGB-D sensor. The color and depth values are registered to one another and therefore allow to extract more accurate information about their similarity. RGB-D data allows to relieve irrelevant and extract robust information to enhance performance and robustness. As the amount of data is huge and particle swarm optimization requires several iterations until convergence is reached, the distance measurement technique has to be fast.

2.5 Summary

The application of PSO for point cloud registration is a challenging problem due to the sensor characteristics and runtime challenges. In this chapter challenges and possible approaches to weaken these constrains were introduced. An outline to possible solutions adressed in this thesis was given considering the specific sensor characteristics and advantages of PSO as an optimization method. In the following chapter state of the art registration methods will be examined. It will be discussed how registration benefits from applying PSO instead of other common techniques.

Chapter 3

3D Registration

In this chapter state of the art approaches for local pose estimation, including point cloud registration systems using a RGB-D sensor and related approaches are discussed. It also gives an overview of different groups of algorithms and compares it to PSO.

3.1 Overview

Registration algorithms obtain **sparse or dense data** as input data. Sparse data is represented by features or landmarks. Dense data registration processes the complete point cloud for optimization. Features or downsampled point clouds are more efficient to compute. One sensor measurement may contain up to 300,000 points. Data reduction to its essential properties is a useful tool to enhance robustness and speed. However, using complete point clouds lead to more accurate solutions. In [RBB09] registration methods are categorized in **global and local approaches**. Genetic or evolutionary optimization techniques are considered as global approaches. Unfortunately, they suffer from high computation time. A popular approach in local optimization is Iterative Closest Point (ICP) [SHT09]. It is called local, because the result of the final optimum is dependent on the initial transformation whereas global approaches are able to find global optima. ICP has the tendency to converge in a local optimum if the datasets are not properly initially aligned [HKH$^+$10]. In [SBB05] registration methods are divided in **coarse and fine registration**. It describes the accuracy of the result of the registration process. Some approaches also combine a coarse registration step, followed by a fine registration step like ICP [SBB05]. For the registration of successive point clouds, algorithms are proposed which examine only **RGB-D or depth data**. For most situations depth data provides sufficient information. It leads to smaller datasets for correspondence estimation. Furthermore, these approaches are invariant to changing lightening conditions. However, in scenarios where depth information is not a reliable similarity indicator, additional appearance-based information is necessary. e.g. wallpapers on a flat wall.

3.2 State of the Art

The approaches presented in this chapter are categorized as correspondence-based or pose-space search methods as outlined in Silva et al. [SBB05]. The more generalized term, search-space-based, is used here instead of pose-space search. A common approach for correspondence-based search is ICP [SHT09]. The following simplified steps are performed iteratively in ICP, if an initial transformation is available: find corresponding points between two sensor measurements and then find a transformation that minimizes the distance iteratively. In this process pairs of correspondences are selected for optimization or marked as outliers.

Correspondence-Based Search

- find correspondences between sensor measurements
- compute transformation to minimize distance

For local pose estimation by population-based approaches, like PSO, a search-space-based search is performed. The steps are processed repeatedly, in reverse-like order compared to ICP: transformations are computed by past experience in order to minimize the distance between two sensor measurements, then the current cost is used to compute future transformations until an optimum is reached.

Search-Space-Based Search

- compute transformations by past experience to minimize cost
- compute cost

The main difference between these concepts is that either correspondences are considered to compute an optimized transformation or correspondences are used to compute a similarity measure, which is then used to guess a new transformation. However, of course both concepts search an optimum transformation in search-space and compute a distance measure based on correspondences. To be more specific, the terms describe how the successive transformation is derived: from correspondences or “guessing” in the search-space based on cost values. Several approaches for algorithms based on correspondence-based and search-space-based search are discussed in the following sections. A detailed insight on similarity measurement techniques, required for both approaches, will be given in Chapter 5.

3.2.1 Correspondence-Based Search

This section is an introduction to point cloud registration techniques which are exploiting RGB-D data for correspondence estimation in frame sets. The approach **RGB-D-ICP** of Henry et al. [HKH$^+$10] applies Scale Invariant Feature Transform (SIFT) as data representation. First, SIFT features are extracted from reference and floating image. Secondly, the 3D locations of the 2D features are extracted from the depth map in order to get point-to-point 3D correspondences. The transformation between the 3D feature correspondences of successive sensor measurements is estimated using RANSAC. This initial feature-based alignment is further improved by ICP using both appearance and depth information. The processing step is described as RGB-D-ICP: The transformation error is iteratively improved by ICP, based on the error of feature-correspondences using euclidean distance and point-to-plane distance of dense point cloud data. The algorithm converges when a minimum error change or a maximum number of iterations is reached. The complete model is represented by small surface patches called surfels. Surfels are a representation method for point clouds, which allows to resample a model to relevant surface information. This processing step is crucial for large model building in order to minimize required data storage. In order to minimize the cumulative error of frame-to-frame alignments, loop-closing and global optimization is executed using a graph structure. The algorithm does not run in real-time: it takes about 150 ms for feature extraction, 80 ms for RANSAC, 500 ms for ICP and 6 seconds for surfel generation. Experiments show improving results using RGB-D-ICP compared to ICP or SIFT alone. The alignment error decreases significantly using graph optimization.

The approach **RGB-D SLAM** by Engelhard et al. [Eng11] is similar to the previous approach: Speeded Up Robust Feature (SURF) are applied for 3D correspondence estimation. Features-correspondences are extracted as described in RGB-D-ICP. The initial transformation is also estimated using RANSAC. The Generalized ICP approach of Segal et al. [SHT09] is applied for refinement. The local correspondence information is added to a pose graph for optimization. Finally, a globally consistent 3D model, represented as a colored point cloud, is the result of the algorithm. The computational complexity makes this approach not suitable for real-time applications.

The approach **Realtime Visual and Point Cloud SLAM** of Fioraio et al. [FK11] performs real-time registration. The input data for generalized-ICP is subsampled and filtered to about 1000 points. Corresponding points are searched via a projection method and rejected if the distance is to large or the angle of the point normals differs too much. The pose is prealigned by performing RANSAC on visual feature points. The global alignment of two frames by bundle-adjustment takes about 60-70 ms.

The approach **KinectFusion** published by Izadi et al. [IKH$^+$11] creates 3D models using full depth data only. There is no feature extraction step required as ICP runs directly on point cloud data computing euclidean point-to-plane distance. A globally consistent volumetric surface-based model of the environment is created and further refined similar to superresolution. It is also extended to non static environments, e.g. to enable users interacting in front of the sensor. Therefore, a reconstruction and interaction mode is available that either builds a model from the complete sensor data or only uses the static background for reconstruction during user interaction. The application areas of this work are focused on segmentation and tracking of objects for augmented reality and physics-based interactions. The algorithm is working in real-time on the GPU.

3.2.2 Search-Space-Based Search

ICP struggles with the requirement of accurate prealignment of sensor data, because it is not robust to outliers from noise or low overlap. Therefore, it easily converges in local optima. To our best knowledge, there is no approach available in literature using a particle swarm technique for RGB-D pose estimation. However, there exist several approaches for search-space-based 3D registration.

For single-slice biomedical images to 3D volume registration Wachowiak et al. [WSZ$^+$04] introduced a method using a hybrid **Particle Swarm Optimization (PSO)** technique. Silva et al. [SBB05] published a Surface Interpenetration Measure (SIM) for range image registration using a **Genetic Algoritm (GA)**. In GA a popoluation of individuals encode solutions by their chromosomes. These are evaluated by a evolutionary procedure: Only the fittest members survive and reproduce in order to create new individuals by crossover and mutation. The algorithm converges at the end of the evolutionary process with the fittest individual containing the best set of chromosomes, i.e. the optimal configuration. PSO has a rather social behavior compared to the "survival of the fittest" concept of GA. An experiment in this publication shows that the registration of about 10.000 3D points takes 5 minutes on a 1.7GHz Pentium IV processor. The algorithm is robust towards low overlap and doesn't need accurate prealignment.

The problems of ICP are also adressed by Lomonosov et al. [LCE06] by extending a Trimmed Iterative Closest Point algorithm (TrICP) with genetic algorithms. It allows to precisely and fully automatically register 3D data with no prealignment. The processing steps are as follows: GA is used for pre-registration of the datasets, then the result is further refined by TrICP.

Other approaches for registration in search-space are only applied in 2D space. In the approach of Jankó et al. [JCE06][JCE07] a genetic algorithm is applied to register a pair of 2D images to an untextured 3D model. Only the texture information is used to build 3D surface models. A visual based SLAM system using modified PSO is presented by Low et al. [LNY10]. It builds feature-based geometric maps in 2D space. Optimization is performed on the feature positions and not on the camera pose itself. A particle represents the location where a feature is observed. When a feature is re-observed, a new particle is added. Therefore, each landmark has its own swarm of particles. As similarity function, the distance to the gravitational center of the swarm is considered. When a swarm converges only one particle is kept to represent the feature location.

3.3 Summary

A wide range of applications emerged with the hype of affordable RGB-D sensors. Since that time, the computer vision community uses this technology increasingly for 3D registration applications. Current approaches for point cloud registration systems using RGB-D sensors still struggle with real-time requirements. However, GPU based approaches allow short computation times. The popular correspondence-based search approach of ICP is commonly used to solve the problem of registration. Anyhow, it suffers from low overlap in sensor data caused by sensor measurements from different field of views and varying discretizations of the surface. It also requires an accurate inital transformation, otherwise it easily converges in local optima. Search-space-based approaches are able to jump out of local optima and do not need an accurate initial alignment. In literature, a combination of both approaches are proposed: a search-space-based optimization for coarse registration and correspondence based for refinement. The research of search-space-based approaches in 3D registration does not seem to be fully exploited. Fast distance measurement techniques and the ability to jump out of local minima provide opportunities for improvement of the registration process.

Chapter 4

Particle Swarm Optimization

This chapter gives an introduction to PSO in Section 4.1 and elucidates a selection of available topologies for PSO in Section 4.2. This is followed by a basic algorithm for PSO and a modified approach which is also applied in this thesis (Section 4.3 and 4.4).

4.1 Introduction

The general concept of PSO was introduced by Eberhart and Shi in [ES04]: The algorithm is initialized by a population of solutions represented as particles. Each particle is located at a random position in search-space. It changes its position according to a velocity term influenced by its own history and its neighbors.

In [KE95] Eberhart and Kennedy describe the developement of PSO from a simulation to an optimization algorithm: The first attempt was to simluate swarms based on nearest neighbor velocity matching and "craziness". Agents were defined as collision-free birds. They were randomly distributed over a pixel grid with 2D velocity vectors. In each iteration step, the velocity of each bird was updated according to its neighbor's velocity. On this way the birds moved synchronously, but never changed their direction once they were synchronized. Therefore, a "craziness"-factor was required, which adds a stochastical component to the birds' movements. This approach created a swarm behavior, but still appeared to be rather artificial. In the successive approach the "craziness" was omitted. A dynamic force was added to the simulation, which globally attracts the whole swarm. However, in real life birds don't know where their swarm's global target position is located. Additionally a "cornfield vector" was introduced, which represents a two-dimensional grid with X and Y pixel coordinates. Each agent stores its best value together with the corresponding 2D position in this grid. This value was called *pbest*. The velocites of the agents were computed by a simple rule: The current velocity is weighted with a negative or positive random value depending on its current position relative to *pbest*. The global best position was represented by *gbest* and available to all agents. Most optimization problems require multiple dimensions and are not linear, e.g. the training of a neural network. It was reasonable to enhance the approach from $2 \times n$ dimensions (x and y position for all n agents) to $D \times n$. In further experiments, the simple rules for velocitiy computations were improved.

The research of PSO is categorized by Eberhart and Shi in the following areas:

- applications
- algorithms
- topologies
- parameters
- merging/combination with other evolutionary computation techniques

Applications for PSO are similar to other evolutionary computation techniques e.g. neural networks, tracking of dynamic systems, reactive power and voltage control for industrial systems, biomedical image registration [KN10][WSZ+04] or staff scheduling [GN09].

PSO algorithms are separated in global and local approaches: In order to find an optimum the global version considers all particles, while the local version is limited to a neighborhood defined by a certain topology. The global version is able to converge fast while the local version is capable of finding good solutions slowly (see Section 4.2 on topologies). The velocity of particles consists of a *cognitive*, a *social* and a *momentum* part. The cognitive component describes the ability of the particle to move through search-space according to its own history combined with a random deviation. It attracts the particle back to its own best position. The social part reflects the influence from the behavior of other particles in the neighborhood. The momentum keeps the particle at its current velocity. A new parameter called *inertia weight* was added to the original version of PSO to control global and local search abilities [SE98]. Another parameter, the so called *constriction coefficient*, was added in order to let the algorithm always converge [Cle02]. However, in the history of PSO modifications of the algorithm and its parameters were presented e.g. by introducing new parameters or removing original parts. The applications of these parameters will be described precisely in Section 4.3 and 4.4.

4.2 Topologies

Topologies influence the effectiveness of a particle heading to an optimum solution. They determine a subset of particles which are *socially influenced* by each other's movement. The best particle in a neighborhood attracts all other particles of this neighborhood to its position. An introduction and comparison of various topology concepts is given in [GN09]. In the global approach *gBest* each particle checks all neighbors, i.e. the whole population in order to find the optimum (see Figure 4.1).

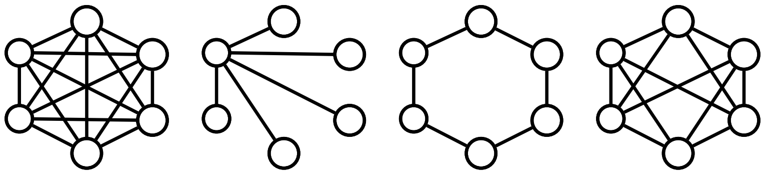
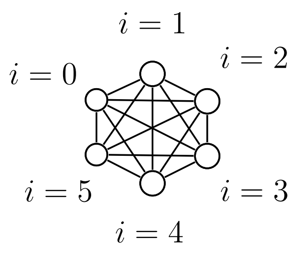


Figure 4.1: Neighborhood topologies: global, wheel, circle and local (k=4)

The global strategy leads to fast solutions, because each particle knows the current global optimum. However, it may converge to a local optimum because no other solutions will be examined when the first global optimum was found. The *lbest* strategy only considers a subset of particles i.e. the particle's neighbors. The parameter k represents the number of neighbors that will be considered. In Figure 4.1 a global topology (k=n-1) with n being the total number of particles in the population, wheel (k=1), a circle (k=2) and a topology with 4 neighbors (k=4) are visualized.

The local approach leads to slower convergence than the global approach, although it is more likely that a global optimum is found. The exploration of search-space is more extensive if subsets of particles optimize their local best position instead of all particles optimizing one global best position. Other promising variations of topologies are described in [GN09]: e.g. a topology changing the size of the neighborhood from k=1 to k=n-1 during execution.

The choice of a suitable social network in order to distribute the information about local best position depends on the problem and according to [GN09] cannot be stated in general. Figures 4.2 and 4.3 show the local best position for global topology and Figure 4.4 and 4.5 for ring topology. During the update step of local best within the swarm, every particle is checked for its neighbors' cost. The particle's neighborhood is defined as including not only other particles, but also the particle itself. The particle's lbest value is updated to the neighbor's current cost and position, if it improves its cost value. In the first update step of *lbest* for particle i = 0, see Figure 4.4 and 4.5, the cost values of its neighbors i = 1 and



i	$f(\boldsymbol{x}_i)$
0	0.3
1	0.4
2	0.2
3	0.3
4	0.4
5	0.1

Figure 4.2: Indices of particles and cost table for local best in global topology

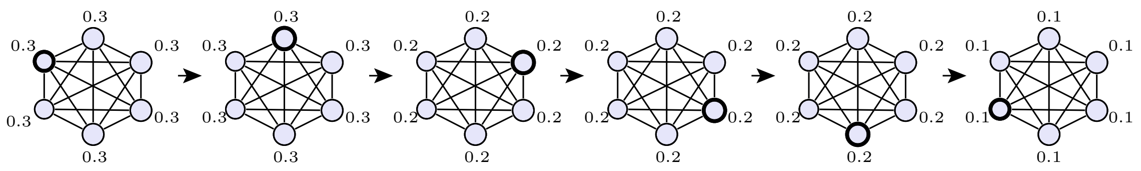
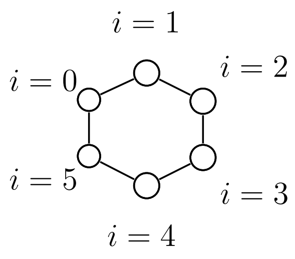


Figure 4.3: Local best in global topology containing 6 particles. The update of lbest is performed by iterating from $i = 0$ to $i = 5$ from left to right. The particles of the active social network are marked with gray background.



i	$f(\boldsymbol{x}_i)$
0	0.3
1	0.4
2	0.2
3	0.3
4	0.4
5	0.1

Figure 4.4: Indices of particles and cost table for local best in ring or circle topology

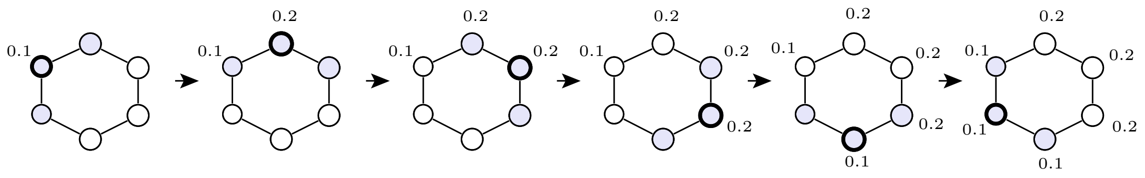


Figure 4.5: Local best in ring or circle topology containing 6 particles. The update of lbest is performed by iterating from $i = 0$ to $i = 5$ from left to right. The particles of the active social network are marked with gray background.

$i = 5$ are checked. The smallest cost among 0.1, 0.3 and 0.4 is given by particle $i = 5$ with a value of 0.1. Therefore, this value is assigned to *lbest* of particle $i = 0$. Next, *lbest* of particle $i = 1$ is updated considering its neighbors $i = 0$ and $i = 2$ and so forth. In general, a high number of neighbors k results in fast convergence of the algorithm, but is accompanied by a spare exploration of the search-space. This behavior leads to convergence in a local optimum solution. This behavior is shown in Figure 4.5: Particles with indices $1, 2, 3$ and $4, 5, 0$ move to the indiviual local best position in their social group. However, in the global topology as in Figure 4.3 only one best position is followed.

4.3 Basic Particle Swarm Optimization

The following section explains the computation steps of basic PSO as presented in [NdMM06]. The processing steps of the algorithm are also available in Listing A.1 in the appendix.

At the beginning the swarm members need to be initialized in search-space in order to give a first clue about possible solutions. The following attributes of each particle i are set:

- current position $\boldsymbol{x}_i$ for each dimension j
- velocity $\boldsymbol{v}_i$ for each dimension j

Position and velocity are initialized with random values of uniform distribution, because there is no other information about possible optima available yet. However, the range of random variables is limited in order to constrain the search-space. During the execution of the algorithm the particles move through search-space in order to find good solutions. The initial position is only used as a starting point to explore the search-space at the beginning. For instance, the dimensions of search-space in 2D image registration are defined by the parameters for position, scale and rotation in case of rigid transformations. However, they always need to be adapted to the problem to be solved.

Each particle i stores the following attributes during search:

- its current position $\boldsymbol{x}_i$ and cost $f(\boldsymbol{x}_i)$
- its current velocity $\boldsymbol{v}_i$
- its best position *pbest* $\boldsymbol{y}_i$ and cost $f(\boldsymbol{y}_i)$ so far
- the best position in the neighborhood *lbest* $\boldsymbol{\bar{y}_i}$ and cost $f(\boldsymbol{\bar{y}_i})$ so far

During one iteration step of PSO, the position of a particle is updated by the following formula:

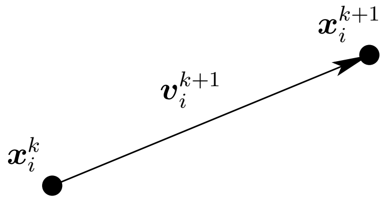


Figure 4.6: Update of position according to Equation 4.1

$$\boldsymbol{x}_i^{k+1} = \boldsymbol{x}_i^k + \boldsymbol{v}_i^{k+1} \tag{4.1}$$

In order to solve this formula, the velocity for the new time step $k+1$ is derived in each dimension j from the following equation:

$$v_{ij}^{k+1} = \omega \underbrace{v_{ij}^k}_{\text{momentum}} + \underbrace{c_1 r_{1j}\{y_{ij}^k - x_{ij}^k\}}_{\text{cognitive component}} + \underbrace{c_2 r_{2j}\{\bar{y}_{ij}^k - x_{ij}^k\}}_{\text{social component}} \tag{4.2}$$

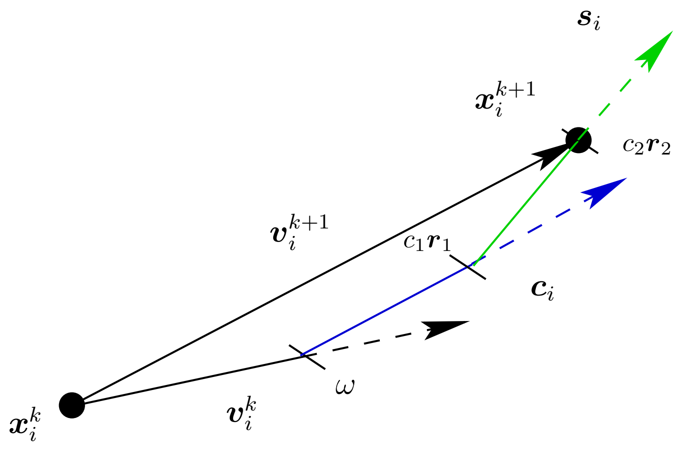


Figure 4.7: Update of velocity according to Equation 4.2.

The variables $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ are random variables in the range $[0, 1]$. They are used as coefficients for the "self-recognition", cognitive component, and "social part" or also called social component. The variable ω is called inertia weight. It controls the amount of influence of the last velocity to the following velocity. If the inertia weight is set too high, the particles' position in search-space can easily "explode". To avoid an exponentialy increasing velocity of the particle, it is possible to add constrains to the search-space such as reinitializing a particle that "got lost". Figure 4.7 shows the update of the velocity according to Equation 4.2 in a 2-dimensional search-space. The velocity v_{ij}^k of the last time step is weighed by ω and then summed up with the social and cognitive component. The vector $\boldsymbol{c}_i$ is part of the cognitive component. It is the vector $\{y_{ij}^k - x_{ij}^k\}$ between the personal best and the current position of particle i. The vector $\boldsymbol{s}_i$ is part of the social component and is the vector $\{\bar{y}_{ij}^k - x_{ij}^k\}$ between the neighborhood best position and the current position of particle i.

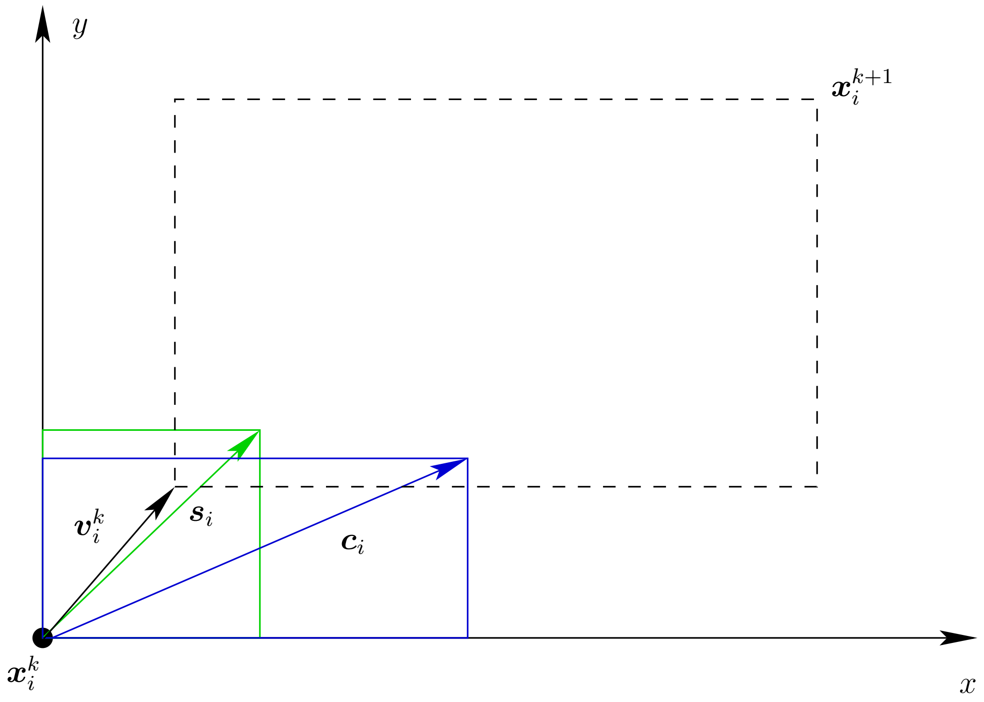


Figure 4.8: Update of position in PSO (adapted from [KN10])

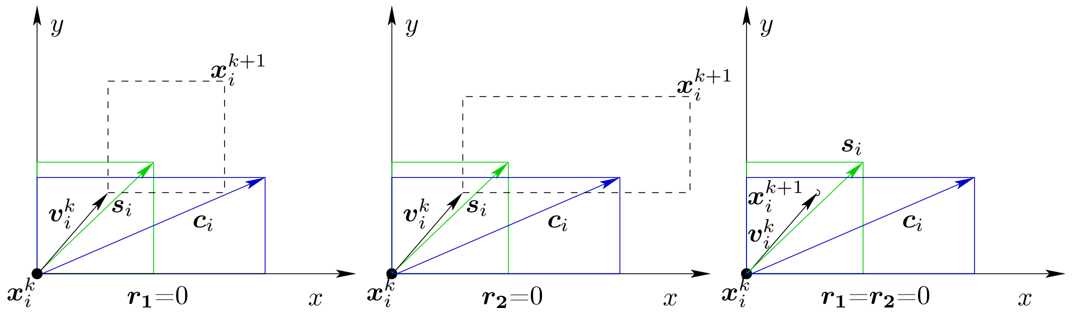


Figure 4.9: Example configurations of cognitive and social component in PSO.

Figure 4.8 shows the geometrical description of the position update considering the influence of cognitive and social component. In this illustration the particle's position $\boldsymbol{x}_i^k$ is at the origin. As $\boldsymbol{c}_i$ and $\boldsymbol{s}_i$ are multiplied by the random variables of uniform distribution $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ in the range $[0, 1]$, the position $\boldsymbol{x}_i^{k+1}$ in the next time step will always be located in the dotted rectangle (when ω, c_1 and c_2 are set to 1). Figure 4.9 shows how the cognitive and social component are influenced by different values of $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$. The amount of influence from social and cognitive component specifies the range of possible next positions $\boldsymbol{x}_i^{k+1}$.

The maximum number of iterations searching for the optimal solution in search-space is limited by different strategies [NdMM06]:

1. fixed maximum number of iteration with improvement
2. fixed maximum number of iterations without improvement
3. threshold for minimum cost

The first strategy is not a good solution in terms of optimizing the search process since a fixed number has to be predefined. The second approach converges when no further improvements of the registration result were made for a predefined number of times. It improves the disadvantage of limited improvement of the first one, but due to the stochastic nature of the algorithm it may leave a local optimum and still be able to find a good solution after more iterations. If a threshold of the objective function is taken as a criterion, it surely converges with the expected quality. However, if no good solution is found the algorithm may run forever. A combination of all criteria may offer the best solution for fast and accurate convergence. Another approach was implemented in this thesis: A delta value and a maximum number of iteration has to be defined by the user. The algorithm converges if the best pose in search-space has a derivation smaller than delta for the given maximum number of iterations. The difference to the second convergence criterion is that the delta is not defined on the cost value, but on the pose change in search-space. Therefore, it allows the user to define a required accuracy for pose estimation independent of the distance metric.

4.4 Modified Particle Swarm Optimization

A modified version of PSO in the context of medical image registration is presented in [KN10]. The following presented PSO modification is used within this thesis. In contrast to the basic PSO, the exploration strategy follows a combination of Gaussian and uniform distribution. This section explains how this new strategy is applied to PSO. The steps of the algorithm are specified in Listing A.2. For initialization all particles are distributed uniformly in search-space as in basic PSO, in order to give a representation of all possible solutions. The following steps are repeated until a stopping condition is reached:

The position of each particle is updated as in basic PSO (see Equation 4.1). In order to solve Equation 4.1 the velocity for time step $k+1$ has to be found for each dimension j. The following equation consists of a momemtum, cognitive and social component combined with a constriction coefficient β:

$$v_{ij}^{k+1} = \beta[\underbrace{v_{ij}^{k}}_{\text{momentum}} + \underbrace{c_1 r_{1j}\{y_{ij}^{k} - x_{ij}^{k}\}}_{\text{cognitive component}} + \underbrace{c_2 r_{2j}\{\bar{y}_{ij}^{k} - x_{ij}^{k}\}]}_{\text{social component}} \tag{4.3}$$

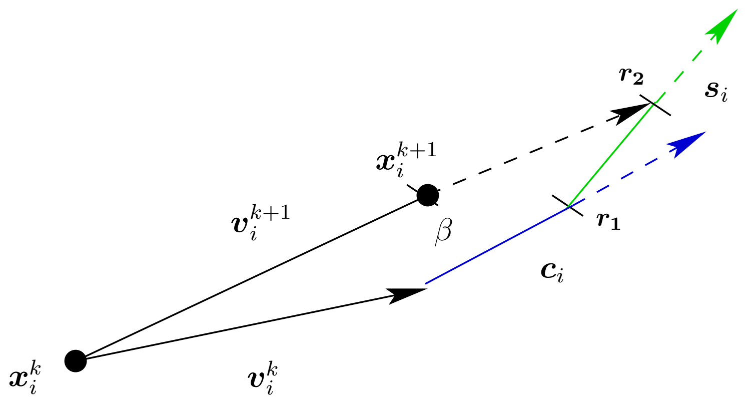


Figure 4.10: Update of velocity according to Equation 4.3

The difference to Equation 4.2 consists in the use of brackets which alters the inertia weight ω to a constriction coefficient β, which is also shown in Figure 4.10. The constriction coefficient was introduced by Clerc et al. in [Cle02].

The constant values c_1 and c_2 set the balance between the impact of cognitive and social component. For each particle i the vectors $\boldsymbol{c}_i$ and $\boldsymbol{s}_i$ are computed as in basic PSO.

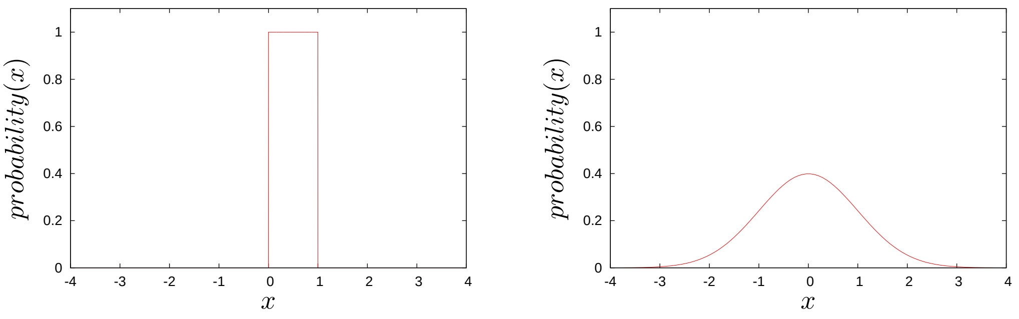


Figure 4.11: Uniform (left) between $[0,1]$ and normal distribution (right) with zero mean and unit variance

Depending on the signs of both vectors the velocity of the particle is updated either by random vectors $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ using a uniform or normal distribution (see Figure 4.11). This is the significant advantage of the proposed modification by Khan et al. [KN10], because using only a uniform distribution often leads to convergence in local minima whereas a normal distribution allows the particle to jump far away. This aims to combine the advantages of both strategies: On the one hand a Gaussian distribution easily converges in local minima, on the other hand it has good exploration characteristics. Meanwhile a uniform distribution is able to exploit the search-space, but lacks exploration abilities.

Figure 4.12 shows the new possible positions of the particle using a normal distribution, which will be located inside the dotted ellipse. In this example the current position of the particle is at the origin, $c_1 = c_2 = 1$ and ω is also set to 1. The new position $\boldsymbol{x}_i^{k+1}$ given by Equation 4.1 is summed up by the current position $\boldsymbol{x}_i^k$ and velocity $\boldsymbol{v}_i^{k+1}$ from Equation 4.3. The velocity results from the multiplication of $\boldsymbol{c}_i$ and $\boldsymbol{s}_i$ with random vectors $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ of a Gaussian distribution, which will always be located in the ellipses with the respective color. Note that this non-uniform probability actually cannot be displayed as values in a certain region and thus are simplified here as an ellipse for visualization purposes.

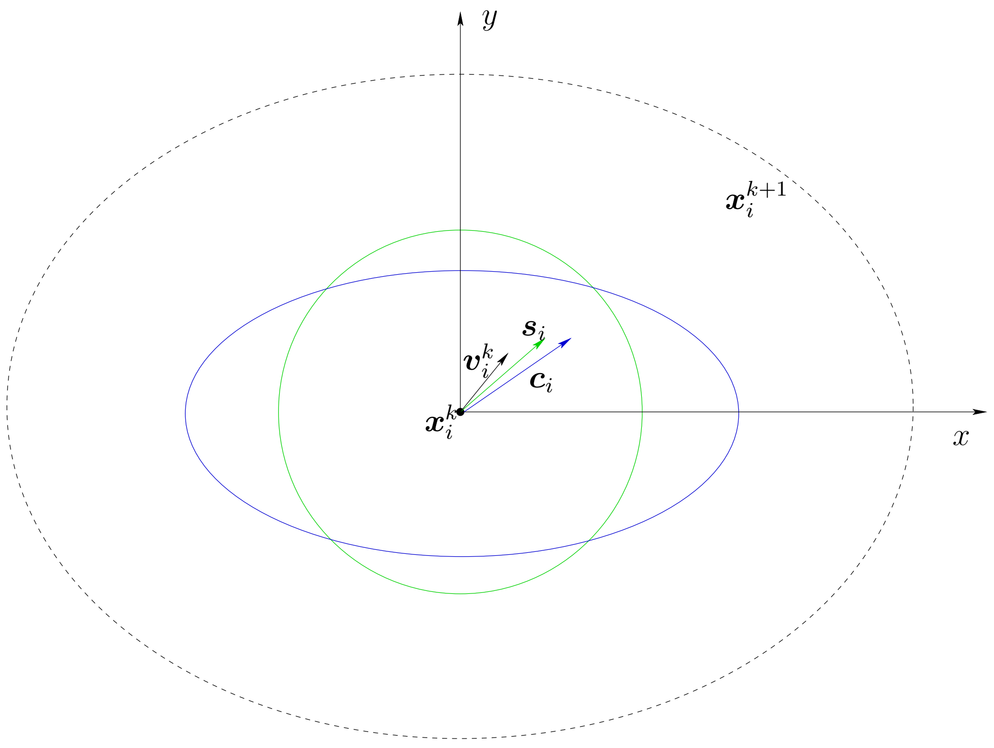


Figure 4.12: Update of position in MPSO (adapted from [KN10]).

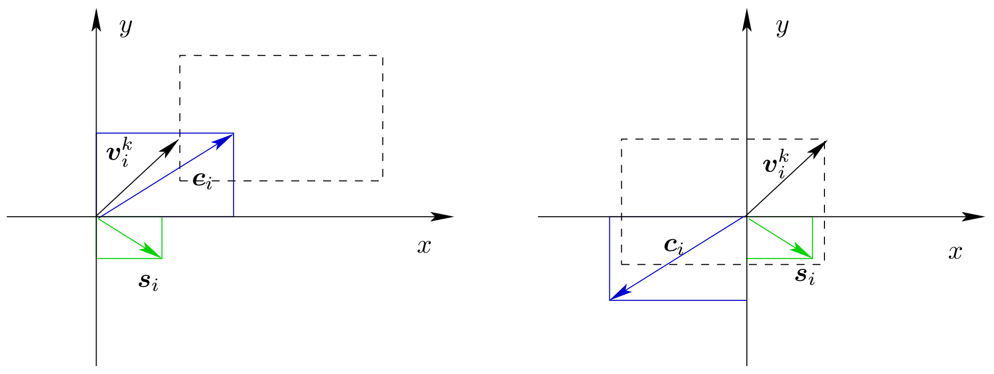


Figure 4.13: Example configurations of cognitive and social component.

Both vectors $\boldsymbol{c}_i$ and $\boldsymbol{s}_i$ have the same sign in all dimensions j if they lie in the same quadrant as in Figure 4.12. In this case a Gaussian distribution is applied, otherwise a uniform distribution as in Figure 4.13. The reason for this distinction is that new positions have the possibility to be located in a larger area compared to the application of a uniform distribution. This behavior is requested if both components agree in their quadrants in order to reach high exploration of search-space. However, if both disagree in signs of dimensions, a uniform distribution constrains the variance of the new position.

Until now the constriction coefficient β was not considered when updating the position of the particles. It is computed by the following formula:

$$\beta = \frac{2\kappa}{|2 - \psi - \sqrt{(\psi(\psi - 4))}|} \tag{4.4}$$

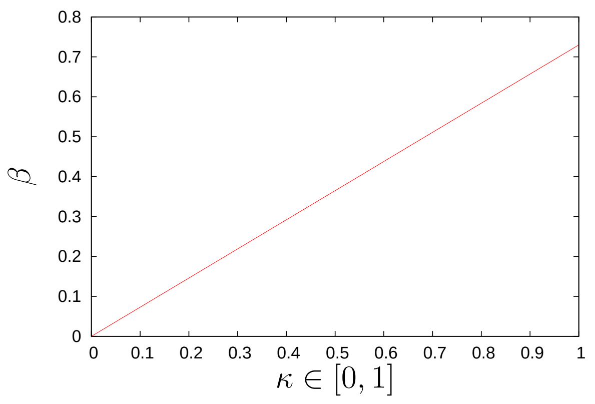


Figure 4.14: Range values of constriction coefficient ($c_1 = c_2 = 2.05$)

The coefficient is defined by the constant $\kappa \in [0,1]$ and $\psi = c_1 + c_2$. The value of κ is often set to 1. A deeper mathematical insight is given in [Cle02]. The values of c_1 and c_2 are often set both equally to 2.05 as described in Khan et al. [KN10]. This results in ψ=4.1, which is larger than 4.0 and thus allows to compute the root of the non-negative term. The value of β controls speed and convergence: large values cause slower and small values lead to faster convergence. The function is linear as depicted in Figure 4.14.

4.5 Summary

This chapter gave an introduction to basic concepts of PSO. The algorithm is used to solve a variety of non-linear optimization problems. In this thesis, the modified PSO approach by [KN10] is applied in order to solve local pose estimation. Contrary to the basic approach, the modified PSO approach uses a normal distribution to move particles in search-space. The successive positions of particles are influenced by either applying a normal or uniform distribution to random variables within the update of the velocity vector. If cognitive and social component are drawn in the same direction, a normal distribution is chosen and if they are aiming in different quadrants, a uniform distribution keeps them from jumping too far away. This behavior leads to exploration and exploitation of search-space at the same time. Exploration is improved by the normal distribution, because it avoids to converge in local optima. Exploitation is achieved by the application of a uniform distribution, which lets particles move in a rather constrained search-space. The main challenge about PSO is to find a suitable similarity measurement in order to apply a cost to each particle. Also, the dimensions of search-space have to be adapted to the problem. These questions will be adressed in detail in the following chapter.

Chapter 5

Similarity Measurement Techniques

Each particle in PSO requires a cost value in order to evaluate its current position. The choice of a cost function is dependent on the kind and quality of input data. The major objective of the thesis at hand is to evaluate similarity measurements in order to efficiently apply them to input data.

5.1 Computational Problems

Various approaches for the comparison of images or point clouds for 2D and 3D registration are available in literature [Gos05] [GHT11]. In this thesis a fast implementation considering appearance and geometry-based features is required. There are scenarios where geometric or appearance-based approaches will fail, if only one approach is applied. For instance: a room with lots of textured items (geometry and appearance features), an empty room or corridor (only geometry features), a planar wall with posters or pictures (only appearance features) or a uniformly colored wall (no features are available and therefore not considered here). Consequently, a combination of features is required for robust registration of point clouds. Different characteristics need to be extracted from the environment and, in case of various feature types, mapped to one cost value.

In this chapter an overview of appropriate techniques is presented to address this problem. In the context of PSO for local pose estimation, one has to struggle with computational costs due to the relatively high number of transformations and similarity computations required until convergence is reached (see Chapter 3). Therefore, in this thesis only distinctive feature points, which implement a reliable representation of their neighborhood, are considered and analyzed by a metric corresponding to their specific properties. In literature, these points are also known as interest points, point landmarks, corner points or control points [Gos05]. These feature points or a combination of them need to be invariant to rotation, translation, scale and viewpoint. Thereby, computational cost is reduced compared to the processing of complete point clouds. Furthermore, the robustness of similarity measurements is increased when irrelevant or erroneous information as described in Chapter 2 is omitted.

The following computation steps are applied in order to compute the similarity between two sensor measurements:

1. extraction of feature points (Section 5.2)
2. search for correspondences (Section 5.3)
3. computation of cost by distance metric(Section 5.4)
4. mapping of cost values (Section 5.5)

These steps will be adressed in the following sections in the presented order.

5.2 Extraction of Feature Points

In the last decade, major research has been reported in computer vision to detect and represent features in 2D images: lines, blobs, regions of interest, interest point detectors and descriptors like SIFT, SURF, SUSAN or Harris corner detector [BGRM07][JKFB06]. These algorithms are also applied in 3D by transforming point clouds to depth images, where each pixel contains depth information encoded as a gray scale value. In the recent years, 3D interest point detectors and descriptors have come in focus of research, e.g. Normal Aligned Radial Feature (NARF) by Steder et al. [SRKB10] or Fast Point Feature Histograms (FPFH) of Rusu et al.[RBB09]. For local pose estimation using PSO, computational expensive detectors and descriptors are not sufficient, due to the computational complexity.

Various techniques for the extraction of representative points will be discussed in the following sections. Not all points $\boldsymbol{p}_i$ for a set of points $P = \{\boldsymbol{p_1} \cdots \boldsymbol{p}_n\}$ are sufficiently distinctive to be detected in subsequent sensor measurements. Therefore, a set of specific points $\bar{P} \in P$ are extracted from the point cloud as feature points, which give a robust and comprehensive representation of the cloud. In the following chapters P will represent the feature points of the point cloud and not the complete cloud itself. The similarity measures are applicable to feature as well as to dense point clouds.

5.2.1 Corners

In computer vision, corner detectors such as Harris can be applied to range or RGB image in order to find distinctive points [Gos05].

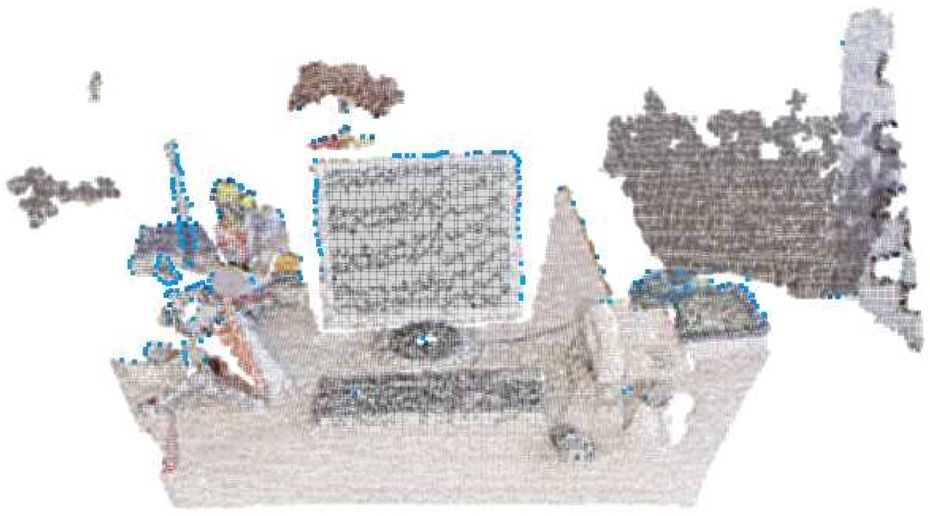


Figure 5.1: Depth Corners (marked as blue points) from dataset [SME+11]

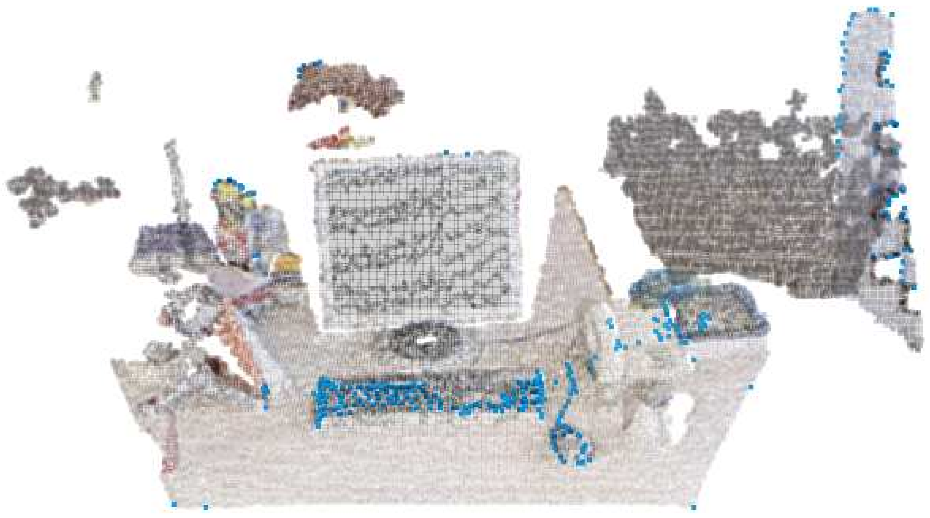


Figure 5.2: Appearance Corners (marked as blue points) from dataset [SME+11]

Figure 5.1 shows an example for detected corners in a depth image, which are reprojected to the point cloud. Appearance corners are extracted using the RGB image as input source. The 2D locations of corners from the RGB data are reprojected to the point cloud. Figure 5.2 shows detected corners in the RGB image.

The computation steps of the Harris corner detector are described in [BGRM07] as follows: The image is divided into patches. A gradient image I_x and I_y for horizontal and vertical direction for each image patch is computed. The gradient matrix $C(x, y)$ for the interest point (x, y) is derived from:

$$C(x,y) = \begin{pmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{pmatrix} \tag{5.1}$$

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 \tag{5.2}$$

The eigenvalues of $C(x, y)$ are λ_1 and λ_2. The variable k is set to a constant value. The auto-correlation function R has maxima where both eigenvalues are high. It is likely that a corner was found at these points. The algorithm is invariant to changes in rotation and translation. However, corner points are detected rather unstable in noisy depth images [SGB10]. In depth images, corners are detected in areas of depth transition. Normals in these areas are not stable to compute, because the neighborhood itself has no stable surface. Therefore, a similarity measure including the surface normal does not lead to feasible results. Also, due to the depth inaccuracy of the sensor, these corner points are not visible in every sensor measurement.

5.2.2 Borders

Borders in range images are extracted by considering the transitions between foreground and background. There are different types of border points: obstacle borders, veil points and shadow borders (see Figure 5.3). Obstacle borders belong to the ”object” itself, while veil and shadow points are artifacts of the sensor's data acquisition for instance due to the limited field of view or occlusion. The object's borders are more distinctive and robust to detect compared to corners. The information about veil or shadow points are also used to filter invalid feature points, which were detected by other methods.

Figure 5.4 shows the detected object borders compared to Figure 5.5, which shows shadow borders and veil points from the same scene. Border points are far more stable than the corner points presented in the previous section. In order to extract borders, the Difference of Gaussians method is adapted from 2D computer vision. However, the detected borders are lying on unstable areas considering their normals. An algorithm for the detection of stable object boundaries in 3D range scans is provided by Steder et al. [SRKB11]. The change of distance in the neighborhood of a point is taken as an indicator for stable borders.

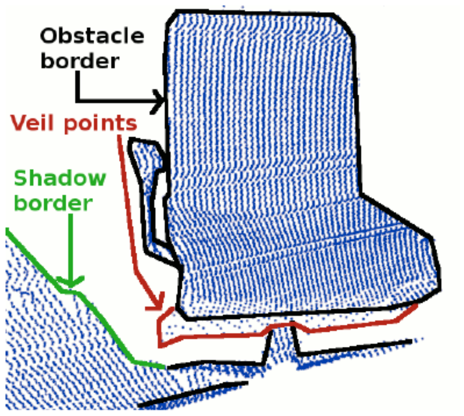


Figure 5.3: Border point types from [SRKB11]

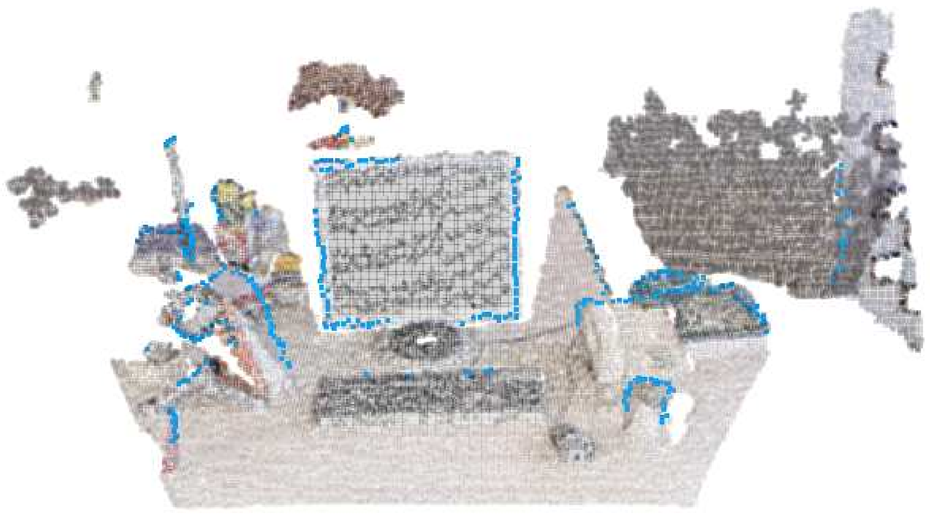


Figure 5.4: Borders of obstacles (marked as blue points) from dataset [SME$^+$11]

5.2.3 Planes

Planar regions are important features of the environment, because they are distinctive and stable to detect. They are view invariant and found in man-made environments. However, they do not provide the possibility of exact localization, because when a point lies anywhere on a plane it can still be an inlier. Figure 5.6 shows the largest detected plane, where the points within the planes are downsampled.

An introduction to plane segmentation methods is given in Holz et al. [HHRB11] and Rusu et al.[RBB09]. Plane detection always requires the detection of normals

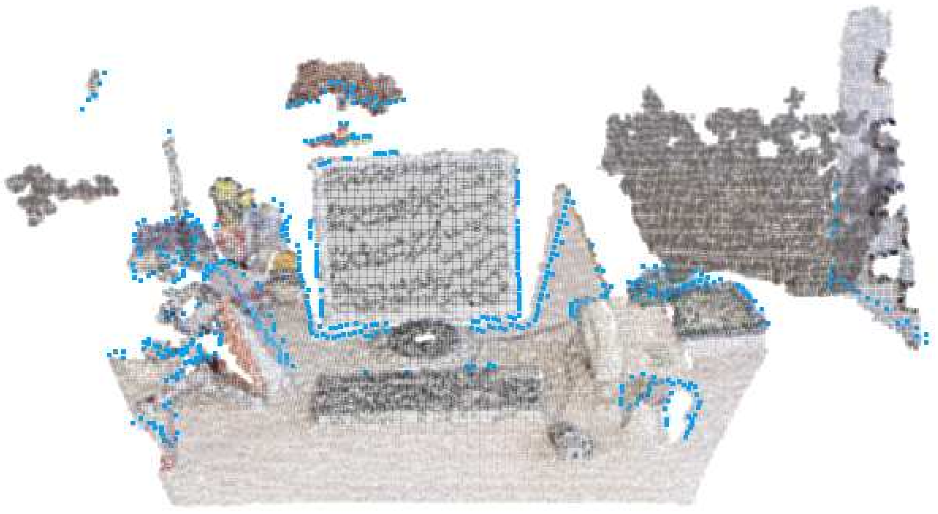


Figure 5.5: Shadow borders and veil points (marked as blue points) from dataset [SME+11]

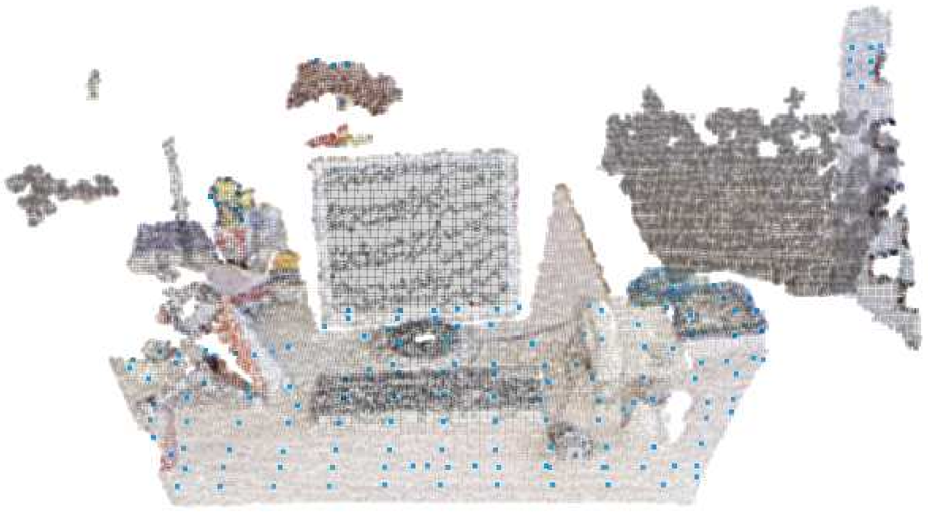


Figure 5.6: Downsampled plane point (marked as blue points) from dataset [SME+11]

in the first step. The normals of a common scene are shown in Figure 5.7. The local surface normals for a point $\boldsymbol{p}_i$ are defined by its neighborhood P_i. This neighborhood is derived from k nearest neighbors or by points within a given radius of point $\boldsymbol{p}_i$. Normals are computed by cross product of this point with its neighbors. Clusters of normals with a similiar pointing direction belong to a common plane.

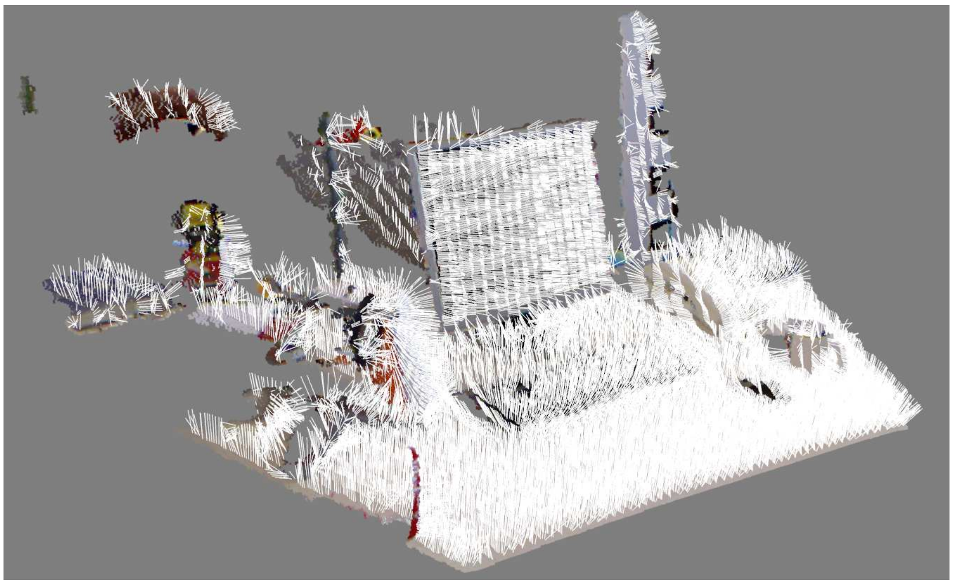


Figure 5.7: Normals of point cloud from dataset[SME$^+$11]

5.3 Search for Correspondences

Correspondences are required in order to decide whether two point clouds are similar. Subsets of these point clouds need to be extracted in order to apply a distance metric on these points. This process is comparable to template matching in 2D computer vision. Outliers should not be considered for similarity estimation as they result from the motion of the camera or other sensor noise. However, the similarity cannot be measured if there is no common subset of points available.

Reference and floating cloud do not necessarily have the same number of points. Additionally, even distinctive points are unlikely to appear again on exactly the same position. Therefore, each point of the first pointset has to find a closest neighbor in the other pointset for correspondence estimation. The definition of the closest neighbor of a point is defined by the selected metric. The concept of neighbor of a given point $\boldsymbol{p}_q$ for a set of points $P^k = \{\boldsymbol{p}_\mathbf{1}^{\boldsymbol{k}} \cdots \boldsymbol{p}_2^k\}$ is given in [Rus09] as

$$\|\boldsymbol{p}_i - \boldsymbol{q}_j\|^2 \leq d$$

with the maximum allowed distance d and $|| \cdot ||$ as a norm. Here the squared euclidean distance is given as an example. The most basic approach to find the closest neighbor is to iterate over both clouds and look for points that fullfill these constrains. However, this approach is too time consuming. In detail: the Kinect has a framerate of 30 Hz where each frame contains a point set with about $300,000$

points. This means that there are $300,000^2$ comparisons required to compute the similarity between two clouds. Fortunately, k-d trees address the problem of finding neighbors quickly by building a k-dimensional search tree to minimize the number of required comparisions.

Figures 5.8 and 5.9 show a schematic example for correspondences between two clouds. The points outside of the field of view should not be considered for similarity estimation. Otherwise the quality of the transformation estimation of the seconds cloud decreases, because a large camera movement with an increasing number of outliers is punished. In Figure 5.8 the clouds are shown and in 5.9 they are transformed by a particle's pose in search-space.

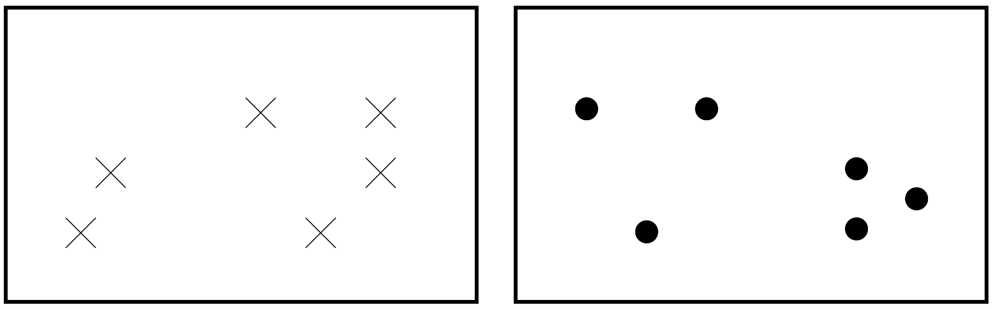


Figure 5.8: Example input datasets for similarity estimation for two clouds

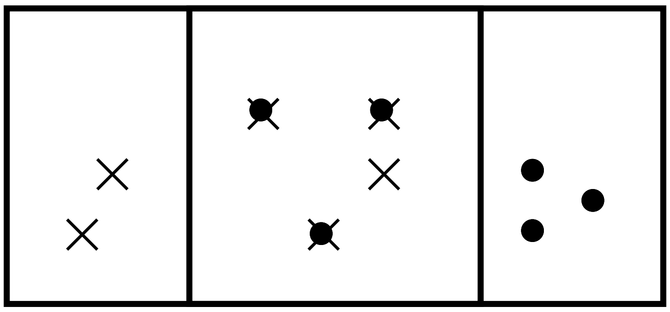


Figure 5.9: Example input datasets aligned according to a particle position.

The points outside of the overlapping area will be dismissed in Figure 5.9. Also in the valid area, the point with no corresponding neighbor will not be considered for similarity estimation.

5.4 Distance Metrics

This section describes distance metrices for cost computation between two point clouds. A normalization step for different distance metrics is described and geometric and appearance-based metrics for point clouds are introduced.

5.4.1 Normalization

Different kinds of error metrics are required in order to measure appearance and geometry feature point similarity individually. In order to use these metrics for PSO, they need to be combined in a common cost value. Therefore, a normalization is performed for all metrics. Also, each metric is normalized individually if it has no fixed boundaries per se.

The cost function f will be introduced as an objective function for point clouds. In order to obtain values in range of $f(P_1, P_2) \in [0, 1]$ the cost function between the point clouds P_1 and P_2 is computed as follows

$$f(P_1, P_2) = \frac{1}{|P_1| \cdot d_{max}} \sum_{i \in P_1} \begin{cases} d(\boldsymbol{p}_i, \boldsymbol{q}_j) & \text{if } \textit{neighbor } \boldsymbol{q}_j \textit{ available} \\ d_{max} & \text{else} \end{cases} \tag{5.3}$$

where $\boldsymbol{p}_i \in P_1$ and $\boldsymbol{q}_j \in P_2$. By iterating over i in P_1, $\boldsymbol{q}_j$ is always defined as the corresponding neighbor to $\boldsymbol{p}_i$. Therefore, the index j will not be redefined for each of the following distance measures in this chapter. The value d_{max} is the maximum distance, which is the maximum cost of the chosen error metric d. In case of euclidean distance it is the maximum distance for the definition of neighborhood. If points $\boldsymbol{p}_i$ without any neighborhood correspondence $\boldsymbol{q}_j$ are not considered by adding a cost value d_{max}, outliers will not receive a penalty. Literally, point clouds with no corresponding points would be evaluated with best cost values. Finally, the cost value is normalized to $[0, 1]$ by the number of points in the point cloud P_1 and d_{max}.

5.4.2 Geometry

This section describes error metrics for geometric features. An overview of further basic distance metrics is given in [Rus09].

Point-to-Point

To estimate the distance between two point clouds the most basic way is to measure the Eudlidiean distance or also called $L2$ norm between the neighboring points $\boldsymbol{p}_i \in P_1$ and $\boldsymbol{q}_j \in P_2$

$$f(P_1, P_2) = \sqrt{\sum_{i=1}^{n} (\boldsymbol{p}_i - \boldsymbol{q}_j)^2} \tag{5.4}$$

However, the squared euclidean distance is often prefered in order to apply a larger cost on points that are far away. This distance measurement technique is also applied as the classical ICP error function [Rus09]. It minimizes the sum of squares distance between corresponding points with the following equation

$$f(P_1, P_2) = \sum_{i=1}^{n} \|\boldsymbol{p}_i - \boldsymbol{q}_j\|^2 \tag{5.5}$$

This error metric is inserted in Equation 5.3 as

$$d(\boldsymbol{p}_i, \boldsymbol{q}_j) = \|\boldsymbol{p}_i - \boldsymbol{q}_j\|^2 \tag{5.6}$$

Point-to-Plane

The point-to-plane or point-to-surface distance is derived from the Hessian normal form for planes and is given as follows

$$f(P_1, P_2) = \sum_{i=1}^{n} \|(\boldsymbol{p}_i - \boldsymbol{q}_j)\cdot\boldsymbol{n}\| \tag{5.7}$$

where $\boldsymbol{n}$ is the normal of point $\boldsymbol{q}_j$. The equation describes the distance between the point $\boldsymbol{p}_i$ and the surface of $\boldsymbol{q}_j$ defined by the distance along the normal $\boldsymbol{n}$. The point-to-plane distance d in $\mathbb{R}^2$ is shown in Figure 5.10 as a point-to-line distance.

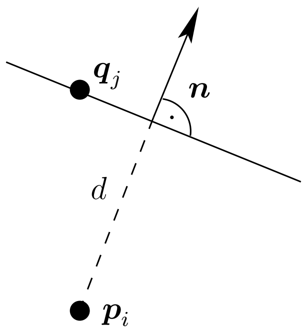


Figure 5.10: Point-to-plane distance in $\mathbb{R}^2$ as a point-to-line distance

The advantage of point-to-plane distance compared to point-to-point distance is that it considers the local geometrical surface around this point. However, normals are not stable at every point in a point cloud (see Figure 5.6). The computation of normals depends on the selection of neighboring points. They are only stable in locally planar areas.

5.4.3 Appearance

Appearance-based distance metrics for point clouds are found in 2D image registration algorithms [Zit03]. Major categories are area-based and feature-based methods. Due to the previous feature point extraction step presented in Section 5.2, there are constrains in the application of matching techniques. For example, if only border feature points were extracted from the original point cloud, it does not make sense to perform an area-based registration methods as there are literally no areas which can be compared. Therefore, the choice of the proper image matching technique is dependent on the type of feature points or regions which were extracted.

In this section, techniques for matching of reference to floating data will be discussed which are also referred as template matching in 2D image registration. In [Gos05] the most common similarity measures are presented: sum of absolute differences, cross-correlation coefficients, moments, Fourier transform coefficients, Mellin transform coefficients, Haar transform coefficients, Walsh-Hadamard transform coefficients, K-S test and mutual information. However, only basic approaches will be presented in this section.

Cross-Correlation

Area-based method like cross-correlation (CC) match intensity values without considering the surrounding texture. The formula for normalized CC (NCC) is given in [BH01]

$$\lambda = \frac{\sum_{x,y}(f(x,y) - \bar{f}_{u,v})\big(t(x-u,y-v) - \bar{t}\big)}{\sqrt{\sum_{x,y}(f(x,y) - \bar{f}_{u,v})^2 \sum_{x,y}\big(t(x-u,y-v) - \bar{t}\big)^2}}$$

where f is the reference image and t is the template or floating image, $\bar{f}_{u,v}$ is the mean of $f(x,y)$ within the template area. The template area is defined by u for the shift in x- and v for y-direction, respectively. The mean of the template image is defined by $\bar{t}$. The normalization by the denominator ensures invariance to changes in brightness or contrast. The range of λ is $[-1,1]$ where 1 expresses the highest similarity, 0 the lowest and -1 indicate a mirrored sensed image. Therefore, values have to be scaled before they are used as a cost or fitness function for PSO.

Normalized cross correlation is also used as a similarity measure for MPSO in medical image registration by [KN10]. In this thesis, for the application of NCC in PSO, the shift u, v for the template is determined by the particle's position. This leads to the following distance function for point clouds P_1 and P_2:

$$f(P_1, P_2) = \frac{\sum_{i,j}(\boldsymbol{p}_i - \bar{P}_1)(\boldsymbol{q}_j - \bar{P}_2)}{\sqrt{\sum_i(\boldsymbol{p}_i - \bar{P}_1)^2 \sum_j (\boldsymbol{q}_j - \bar{P}_2)^2}}$$

if all points in P_1 and P_2 have neighbors $\boldsymbol{p}_i$ and $\boldsymbol{q}_j$. The drawback of techniques like template matching in image registration is the diffculty of finding the right template window. This optimization process is accomplished by PSO. The algorithm is robust towards slight changes in rotation and scaling, but not towards more complex transformations of the image. Also, intensity changes by noise or illumination may alter the results.

Intensity distance

A faster but less accurate approach than NCC is the simple sum of absolute differences of intensity values [Gos05]

$$f(P_1, P_2) = \sum_{i,j}(\boldsymbol{p}_i - \boldsymbol{q}_j)$$

if all points in P_1 and P_2 have neighbors $\boldsymbol{p}_i$ and $\boldsymbol{q}_j$. It returns a cost value, where smaller values are interpreted as a higher similarity between the input data. The cost value has to be normalized as in Equation 5.3.

Color distance

Color similarity provides more distinctive information than just intensity measures. In [SC97] color distance in HSV color space is proposed. Compared to RGB it represents color in hue, saturation and intensity. The transformation from RGB to HSV color space is described in [SC96]. In this thesis, the similarity between two colors $m_i = (h_i, s_i, v_i)$ and $m_j = (h_j, s_j, v_j)$ is given by

$$a_{i,j} = 1 - \frac{1}{\sqrt{5}}[(v_i - v_j)^2 + (s_i \cos h_i - s_j \cos h_j)^2 + (s_i \sin h_i - s_j \sin h_j)^2]^{\frac{1}{2}}$$

The distance of the cylindrical HSV color space between these two colors is measured. If regions instead of point-to-point distances are examined, color histograms can be applied to represent distributions of colors.

5.5 Mapping of Cost Values

The final step of similarity estimation is the mapping of cost values from different domains to one value. However, the combination of geometric and appearance-based features has not yet been examined intensively [HKH+10]. Appearance information alone may lead to wrong correspondences, such as shadows on unicolored walls. The combination of depth and appearance information is the key to robust measurements. A simple approach is to compute the mean of all distance measures and weigh them equally as in the following equation

$$f(P_1, P_2) = \frac{1}{n} \sum_{i=1}^{n} f_i(P_1, P_2) \tag{5.8}$$

where i is the index of the distance metric applied by function f.

The RGBD-ICP approach by Henry et al. [HKH+10] combines appearance and geometry information by the following steps: First, visual features of point clouds P_1 and P_2 are extracted. The corresponding features are initialy aligned. Then, an ICP loop on dense point cloud data is performed. The goal is to find the optimal transformation $\boldsymbol{t}^*$ between P_1 and P_2 by minimizing the distance between feature point correspondences A_f and dense points A_d. A balancing coefficient α is used to balance the influence of appearance and geometry. The geometric distance of the point clouds is computed by a point-to-plane measure. However, in this thesis no features such as SIFT or SURF are applied, which are used in [HKH+10] as an initial alignment. The reason is that the computation of feature descriptors and correspondence estimation are too time consuming for the application in PSO. Equation 5.8 provides a tool for mapping of several distance measures. However, it does not solve the problem that distance values of various metrics are hard to compare even though they are normalized to the same range. Though, the influence of different domains is essential to the quality of similarity estimation.

5.6 Summary

In this chapter an approach for cost computation between reference and floating data of point clouds was introduced. First, methods for feature point extraction in various domains were presented. This was followed by a definition of neighborhood and an introduction to distance metrics. Finally, methods for mapping of cost values from various domains to one cost value $\in [0, 1]$ was presented.

Chapter 6

Implementation of Local Pose Estimation System

The implementation of the system for local pose estimation between two sensor frames is based on the framework "Robot Operating System" (ROS)[1] and is written in `C++`. In this thesis, the ROS release "electric" is utilized. ROS was introduced by Morgan et al. [QGC+]. It provides a modular software architecture, libraries, hardware drivers and tools for visualization. ROS is a distributed system of processes, which aims at providing reusable code for robotic research and development. The code is organized in stacks, which are collections of packages. Nodes are executable programs and wrapped up in packages. The communication between nodes is realized by messages. Each message has a type, e.g. for laser data, and a unique topic name, for example "top-laser". If a node wants to subscribe to a certain topic it has to provide a callback function for the message type. The distributed nature of the system allows to add external packages. They are able to exchange data between nodes, which are implemented within a package. The configuration parameters of ROS-nodes can be altered during runtime.

The Point Cloud Library (PCL) is used for 3D point cloud processing (version 1.3). Tasks like filtering, segmentation, surface reconstruction and model fitting of point clouds are provided by the large scale open source project[2]. In this thesis, the concept of the implementation aims at a modular object oriented structure, which should be easy to understand and well documented. The code is distributed in independent nodes or libraries in order to simplify extension and exchange of components. The formatting and documentation follows the ROS `C++` Style Guide[3].

[1]`http://www.ros.org`
[2]`http://pointclouds.org/`
[3]`http://www.ros.org/wiki/CppStyleGuide`

This chapter gives an overview of the implementation of a PSO libary and its application for RGB-D registration. Section 6.1 discusses the employment of the Kinect sensor and its simulation for test environments. The local pose estimation system is introduced in Section 6.2. The implementation of PSO is described in Section 6.3, followed by similarity measures in Section 6.4. Finally, Section 6.5 gives a summary of this implementation.

6.1 Hardware

The initial step for the registration of RGB-D data is sensor data aquisition. Here, the data is provided through message-passing from the ROS node *openni_node*[4]. The raw sensor RGB and depth data needs to be processed in a calibration step in order to create registered RGB-D point clouds. For evaluation purposes the simulation of sensor data is required in order to make results reproducible. Therefore, logfiles of point cloud data are recorded in order to replace the sensor.

6.1.1 Sensor Calibration

The Kinect sensor has a factory calibration, however in order to obtain more accurate data a manual calibration can be performed by the *camera_calibration*[5] package. This package provides a calibration tool using a checkerboard as in the default OpenCV camera calibration. The calibration data is stored in *yaml* file format and can be loaded by the openni driver using the reference to this camera calibration file.

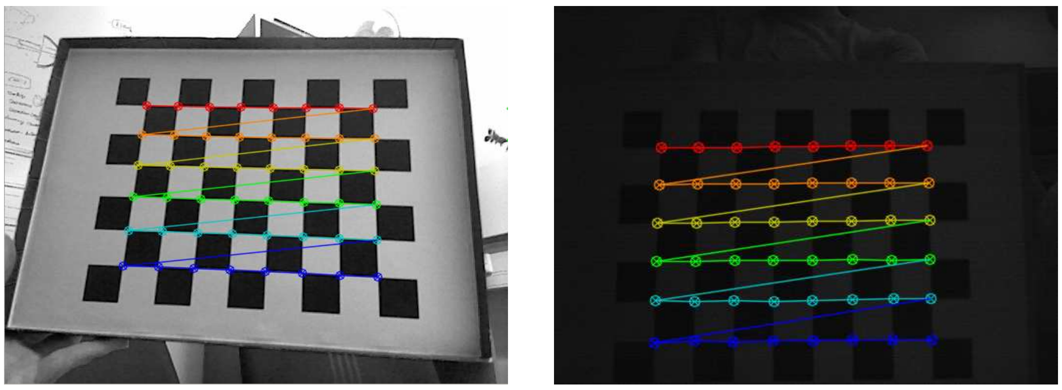


Figure 6.1: Calibration of color (left) and depth data (right)

[4]http://www.ros.org/wiki/openni_camera
[5]http://www.ros.org/wiki/openni_camera/calibration

The calibration tool locates the checkerboard with 8×6 squares of size 2.45 cm in the given camera image and take samples on different x,y positions, size and skews of the board. An image of the detected checkerboard during the calibration data aquisition is depicted in Figure 6.1. In order to get a suitable IR image, the structured light projected from the sensor's IR light source needs to be covered. The detection of the calibration pattern can be improved with an external infrared light source like sunlight or a plastic sheet on the sensor's IR light source in order to diffuse the structured pattern. The simultaneous calibration of depth and color data is not available, because the sensor allows only the aquisition of color or depth information at a time. Therefore the data is not absolutely synchronuous and cannot be calibrated at once.

6.1.2 Logging and Playback

ROS provides tools for saving and playing back message data in "bags". Bagfiles are able to store sensor data for the requested message topics. With bagfiles repeatable tests of algorithms are possible as will be shown in the Chapter 7 on evaluation.

6.2 Local Pose Estimation System

The local pose estimation system is based on a tree of coordinate systems with a fixed basis and a moving camera coordinate system. The camera coordinate system is surrounded by particles searching for the successive camera position. The particle movement is implemented by a wrapper for PSO. The software is started in the user GUI or via command line and configured in launch-files. For visualization the ROS visualization tool *rviz* is applied in order to let the user trace the swarm's movement and the construction of the point cloud model.

In Section 6.2.1 the different coordinate frames and their purposes are introduced. The wrapper for PSO will be explained in Section 6.2.2. A control GUI is presented in Section 6.2.3, followed by the extensive visualization techniques for the internal behavior of the algorithms in Section 6.2.4.

6.2.1 Coordinate Frames

Coordinate systems are managed by using the *tf* package[6]. It allows to listen and broadcast transformations of a tree structure over time. An overview of all coordinate frames in a common hierarchy is shown in Figure 6.2. The basis is the world coordinate system, which is always located at a fixed position and serves as a static

[6]`http://www.ros.org/wiki/tf`

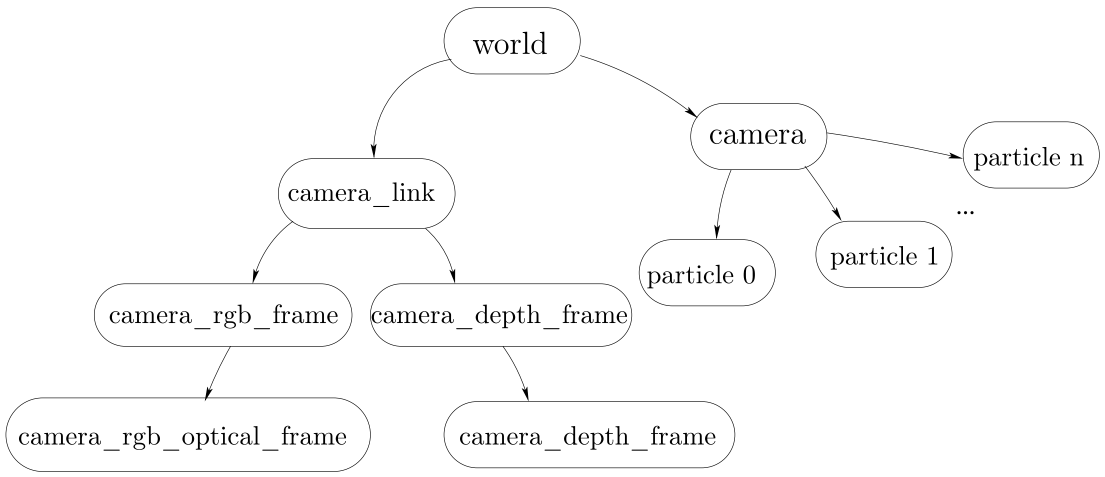


Figure 6.2: Hierarchical view of coordiante frames of own approach

root of all other systems. Therefore, the registered point clouds will be stored in this global coordinate system. The left side of the tree, containing the *camera_ link* as a root node, is broadcasted by the Kinect ROS node. RGB-D point clouds with depth registration enabled are available in the *camera_ rgb_ optical_frame*. The other branch of the tree is created for the local pose estimation system. The camera frame represents the current position of the camera relative to the fixed world coordinate system. There are n particle frames that are transformed using the current camera position as an origin. The camera coordinate sytem is right handed and follows the coordinate frame conventions defined in[7].

6.2.2 Wrapper for Particle Swarm Optimization

Particle swarm optimization is wrapped by an abstract class called PSO, which allows to derive different implementations of PSO. It implements the import of configuration parameters and the publication of swarm, camera and point cloud information for visualization. However, its most important task is to initialize the particle swarm and schedule the optimization process in parallel threads. Figure 6.3 shows a schematic overview of classes containing only the most important properties.

In this thesis two approaches are implemented: the basic approach is designed to process downsampled point clouds for similarity estimation. The feature based approach extracts feature points as described in Chapter 5. The classes, which are derived from the abstract PSO class, need to implement the virtual function *processParticle* in order to assign each particle a cost value. The actual optimiza-

[7]ROS wiki for coordinate frame conventions `http://www.ros.org/reps/rep-0103.html`

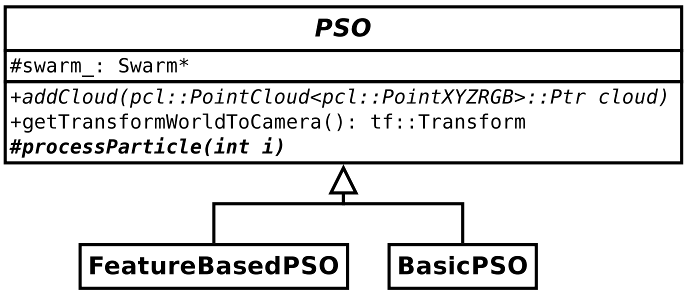

PSO
#swarm_: Swarm*
+addCloud(pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud)
+getTransformWorldToCamera(): tf::Transform
#processParticle(int i)
FeatureBasedPSO
BasicPSO

Figure 6.3: Wrapper for Particle Swarm Optimization

tion is performed in the PSO class. The derived classes implement the interface between the PSO library for optimization, the similarity estimation measures for RGB-D data and the perception of point cloud data. The implementation of PSO will be described in detail in Section 6.3 followed by the similarity measurement techniques in Section 6.4.

6.2.3 User Interface

A basic user interface was developed to start different scenarios without the command line (see Figure 6.4). It is implemented as a seperate package called *control_gui*.

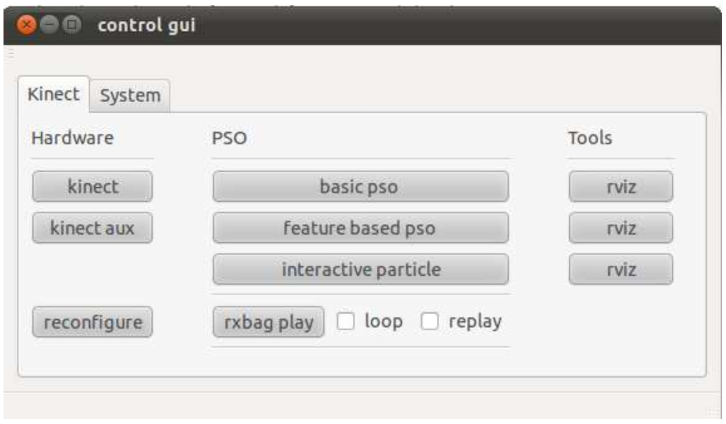
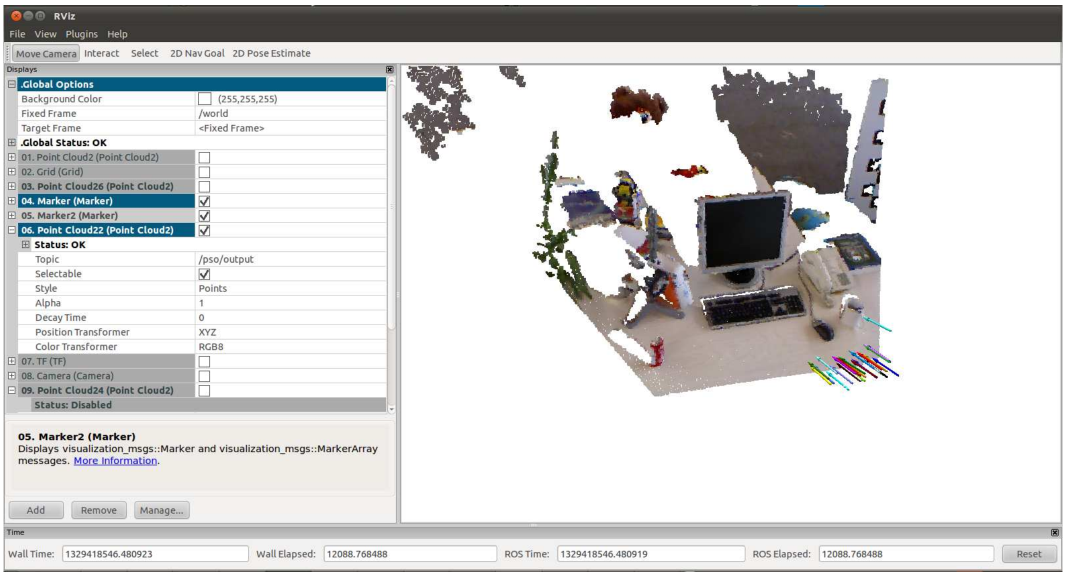


Figure 6.4: Control GUI as a user interface to start nodes.

Various configurations of the ROS visualization tool *rviz* are available for basic or feature-based PSO defined by the corresponding *rviz* buttons. Also debugging by controlling interactive particles is possible. More details will be given in Section 6.4.2. Bagfiles are started and played in a loop or replayed without having to choose the file again by the *rxbag play* button. Loop means that a file is autmatically replayed when it is finished. On the contrary, replay does not restart a file automatically. Only when the *rxbag play* button is pushed the file dialog is omitted and the last file is selected automatically for playback.

6.2.4 Visualization

For visualization *rviz* provides extensive tools for sensor data visualization and possibilities for configuration. Figure 6.5 shows the user interface with subscriptions to various message types like point clouds, coordinate frames, camera images and markers on the left and the visualization of these data structures on the right.



.Global Options	
Background Color	☐ (255,255,255)
Fixed Frame	/world
Target Frame	<Fixed Frame>
.Global Status: OK	
01. Point Cloud2 (Point Cloud2)	☐
02. Grid (Grid)	☐
03. Point Cloud26 (Point Cloud2)	☐
04. Marker (Marker)	✓
05. Marker2 (Marker)	✓
06. Point Cloud22 (Point Cloud2)	✓
Status: OK	
Topic	/pso/output
Selectable	✓
Style	Points
Alpha	1
Decay Time	0
Position Transformer	XYZ
Color Transformer	RGB8
07. TF (TF)	☐
08. Camera (Camera)	☐
09. Point Cloud24 (Point Cloud2)	☐
Status: Disabled	

05. Marker2 (Marker)
Displays visualization_msgs::Marker and visualization_msgs::MarkerArray messages. More Information.

Wall Time: 1329418546.480923 Wall Elapsed: 12088.768488 ROS Time: 1329418546.480919 ROS Elapsed: 12088.768488

Figure 6.5: Visualization of local pose estimation using *rviz*: Subscriptions to various message types are located on the left. They are configured directly and enabled or disabled through checkboxes. The visualization of data structures is shown in the interactive 3D view on the right. Its coordinate system depends on the selected root coordinate frame in the configuration bar.

Particles are literaly moving through search space while a point cloud model is built from incoming sensor measurements. The user is able to navigate within the 3D view. Reference and floating point clouds as well as feature points can be examined during the registration process. The visualization is an essential tool for debugging and demonstration of the internal workings of the algorithm like the movement and distributions of particles and the camera's estimated trajectory. The result of successive pose estimation is given in Figure 6.6. The estimated trajectory of the camera is compared to ground truth data in the evaluation of Chapter 7.

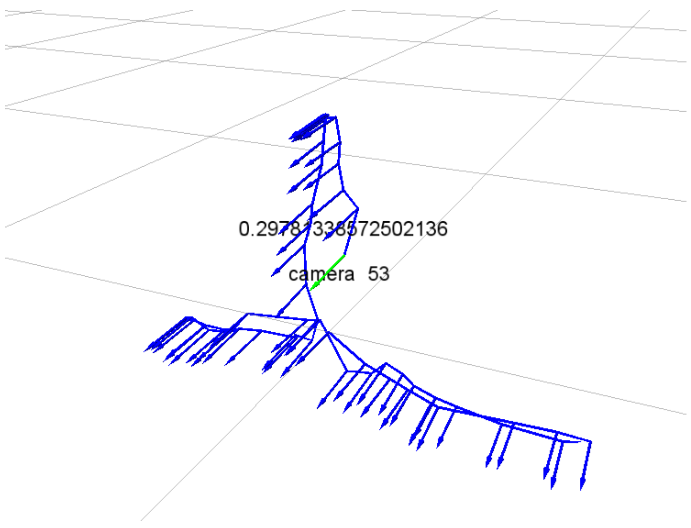


Figure 6.6: Visualization of the estimated trajectory. Camera positions are visualized as blue arrows with the latest pose marked in green. The successive camera poses are connected by a blue line. The number of the registered poses and the current cost of the best particle is printed in digits.

Particles moving through search space can be observed in two different visualization modes. If the ID of each particle is represented by a unique color, particles are easily tracked visually (see Figure 6.7). It enables the user to understand each particle's behavior while exploring the seach space. Different distributions and velocities can be examined directly from looking at the visualization. It is recommended to apply the current cost value as a color intensity, if the distribution of cost values in search space is in the center of interest. In Figure 6.7 on the right, the particles were initially located at the last known camera position. However, their color turns dark for high costs, because the camera has already moved away from the last known position. The swarm starts to explore the search space originating from the last known camera position. The particles' color gets increasingy bright as the swarm is drawn to the subsequent estimated pose.

PSO is configured using different types of topologies. Here, these social networks are visualized as lines connecting two particles with each other. In Figure 6.8 a swarm with global topology is pictured. This topology defines a connection from each particle to all other particles, which is shown in the global topology pattern as well as in the visualization of the implementation. Furthermore, Figure 6.9 and 6.10 demonstrate ring and wheel topology, respectively. Ring topology has two connection lines for two neighbors, but wheel topology has only one. Therefore, in global topology the swarm is much more drawn together as an optimum is propagated fastly among the swarm's members. In ring and even more in wheel topology

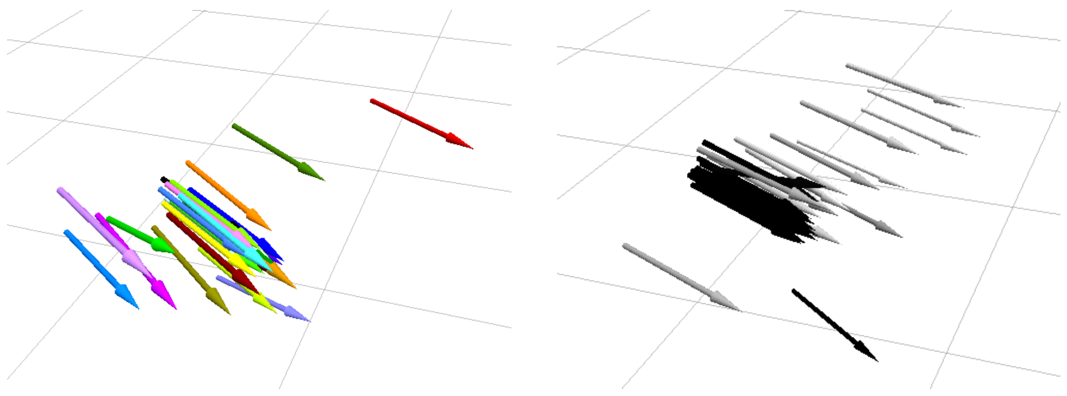


Figure 6.7: Visualization of particles with color representing the ID (left) and intensity for cost values (right), where lower cost equal brighter color.

there are less "communication lines" available and therefore the particles explore the search space rather individually. Experience shows that global topology on the one hand converges faster than other topologies with less neighbors. On the other hand, the exploration of search space is worse, which leads to convergence in local optima. Taking into account the other extreme using wheel topology with only one neighbor, the communication within the swarm is restricted and convergence is time consuming. Wheel topology has shown to be a good tradeoff between exploration and exploitation of search space. This means that particles have the freedom to move around in search space (exploration) and in the meantime are drawn together in regions of optima (exploitation). Thus, ring topology is used as a default topology in this implementation.

6.3 Particle Swarm Optimization

The implementation of PSO is adapted from Khan et al. [KN10] as a generic PSO library. Here, generic means that there is a modular component in the framework which is adaptable to optimize also other problems than local pose estimation. It provides a swarm with a dynamic number of dimensions of search space. During the construction of the swarm various parameters are given which define its search behavior. These parameters may be changed depending on the problem to be solved, e.g. pose range and delta need to be defined depending on the expected distance between successive sensor measurements. The configuration parameters of the swarm are shown in Table 6.1.

The swarm converges if one of the predefined stopping condition are met. In this thesis, the convergence criteria of Table 6.2 were implemented. The default convergence criteria consists of a maximum number of iterations combined with

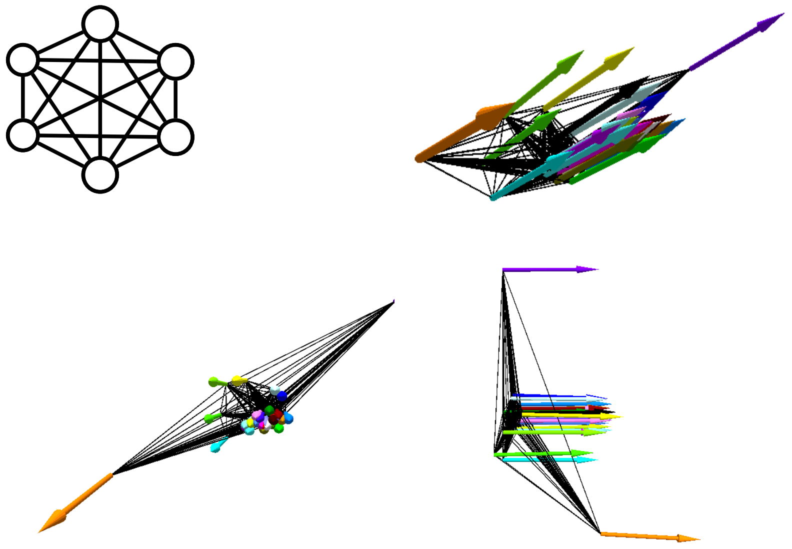


Figure 6.8: Visualization of a swarm with global topology and its schematic pattern

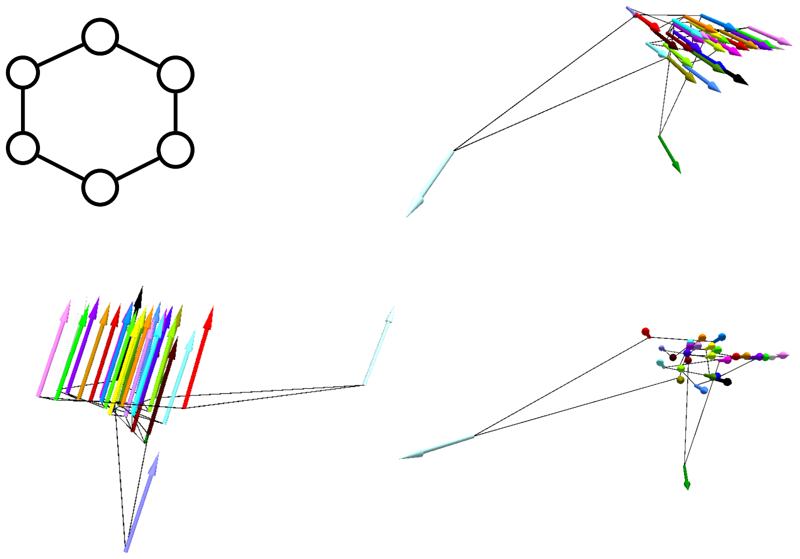


Figure 6.9: Visualization of a swarm with ring topology and its schematic pattern

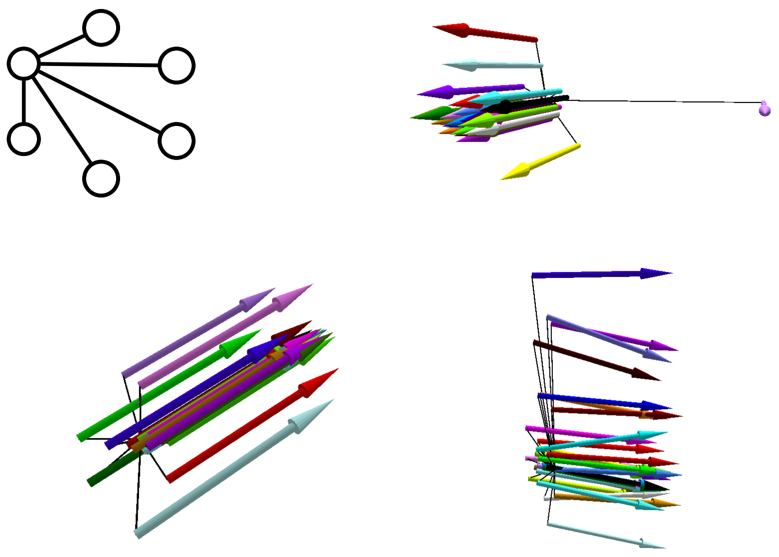


Figure 6.10: Visualization of a swarm with wheel topology and its schematic pattern

topology	global, ring or wheel topology; default: ring
size of the swarm	needs to have at least 3 members
c_1 and c_2	weighing factors for cognitive and social component; default: 2.05
κ	influences speed of convergence; default: 1
σ	sigma of normal distribution for velocity
pose range	range of initial pose of each particle
pose delta	defines the range where a particle needs to be reset
velocity	initial velocity of each particle

Table 6.1: Configuration parameters of a swarm in the PSO library

min error threshold	minimum cost value for convergence
max iteration (pose delta)	number of iterations with given pose delta
max iterations (zero delta)	number of iterations without improvement
max iterations	maximum number of iterations

Table 6.2: Convergence criteria of a swarm in the PSO library

the maximum number of iteration with a pose delta. Thus, a value for accuracy is defined and convergence time is limited by a maximum number of iterations. The iteration steps of the optimization algorithm are listed in appendix A.2.

6.3.1 Class Hierarchy

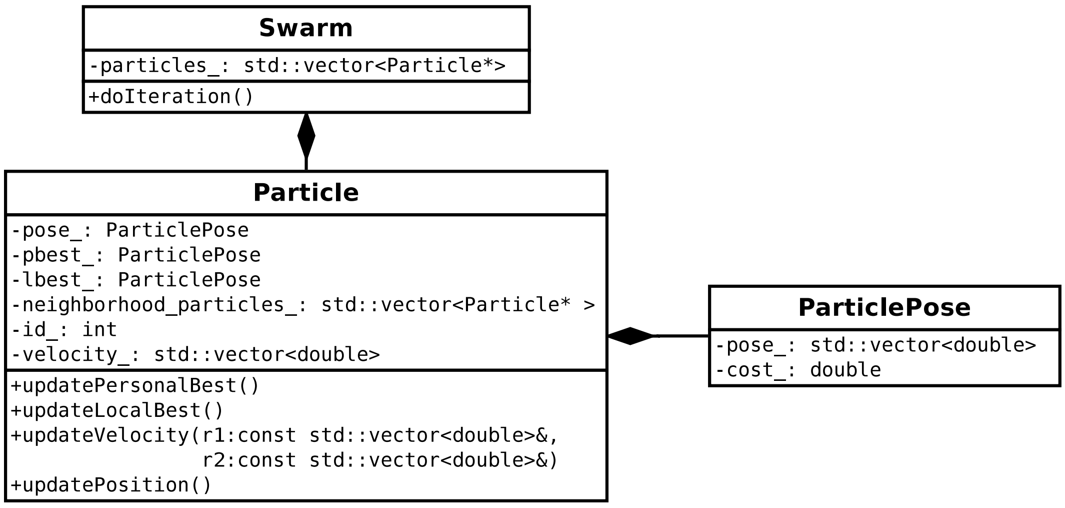

Swarm
-particles_: std::vector<Particle*>
+doIteration()
Particle
-pose_: ParticlePose
-pbest_: ParticlePose
-lbest_: ParticlePose
-neighborhood_particles_: std::vector<Particle* >
-id_: int
-velocity_: std::vector<double>
+updatePersonalBest()
+updateLocalBest()
+updateVelocity(r1:const std::vector<double>&, r2:const std::vector<double>&)
+updatePosition()
ParticlePose
-pose_: std::vector<double>
-cost_: double

Figure 6.11: Class hierarchy of PSO library

The structure of the implementation of PSO is shown in Figure 6.11. The diagram shows a simplified version of the implementation (some functions and members are omitted for clearness). The *Swarm* class operates as the manager of all particles. The swarm's members, topology and behavior are created and controled from here. During optimization the iteration steps are performed by the swarm until convergence is reached. Each member in the swarm is represented as an instance of the *Particle* class. It stores the current position, its best position so far and best position in the neighborhood as an instance of *ParticlePose*. The *ParticlePose* class holds position and cost data. It is applied to represent $\boldsymbol{x}_i$ and cost $f(\boldsymbol{x}_i)$, $\boldsymbol{y}_i$ and $f(\boldsymbol{y}_i)$ or $\bar{\boldsymbol{y}}_{\boldsymbol{i}}$ and $f(\bar{\boldsymbol{y}}_{\boldsymbol{i}})$ of a particle. The topology of a swarm is implemented as a set of references to other particles in each particle. Each particle has a unique ID, which is required to identify each particles for visualization.

The Particle class implements the essential steps of the MPSO algorithm: *updateVelocity* computes the particle's velocity considering previous velocity, cognitive and social component and random vectors r_1 for the cognitive component r_2 for the social component. In order to update the particle's position, *updatePosition* considers the previous position and new velocity of the particle. The variable personal best $\boldsymbol{y}_i$ and cost $f(\boldsymbol{y}_i)$ are updated by *updatePersonalBest* and local best $\bar{\boldsymbol{y}}_{\boldsymbol{i}}$ and cost $f(\bar{\boldsymbol{y}}_{\boldsymbol{i}})$ by *updateLocalBest*. These update functions are called when executing the *doIteration* function of the *Swarm* class. First, personal $\boldsymbol{y}_i$ and local best $\bar{\boldsymbol{y}}_{\boldsymbol{i}}$ are updated together with their cost. Then, new velocities for all particles are computed according to the direction of vectors for personal $\boldsymbol{y}_i$ and local best pose $\bar{\boldsymbol{y}}_{\boldsymbol{i}}$ originating from current pose $\boldsymbol{x}_i$. Either a uniform or normal distribution

Domain	Properties
Obstacle borders	RangeImageBorderExtractor from PCL, see Section 5.2.2
Planes	SACSegmentation from PCL, see Section 5.2.3
Depth corners	goodFeaturesToTrack[8] from OpenCV, see Section 5.2.1
Visual corners	goodFeaturesToTrack[8] from OpenCV, see Section 5.2.1
Clusters	EuclideanClusterExtraction from PCL [Rus09]

Table 6.3: Domains of feature clouds

Similiarity measure	Properties
Point-to-Point	Squared euclidean distance, see Section 5.4.2
Point-to-Plane	Point-to-plane distance between point clouds, see Section 5.4.2
NCC	Normalized Cross correlation of intensity values, see Section 5.4.3
Euclidean distance color	Euclidean distance between RGB values
SAD	Normalized sum of absolute distances between intensity values, see Section 5.4.3
HSV	Distance in HSV color space, see Section 5.4.3

Table 6.4: Similarity measures

is applied to the velocity update depending on the directions of personal and local best position. Finally, the position is updated and the swarm's members are sorted according to cost $f(\bar{\boldsymbol{y}}_{\boldsymbol{i}})$ of personal best $\boldsymbol{y}_i$ in descending order.

6.4 Similarity Measures

For feature-based PSO, various domains for the representation of point clouds were implemented. An overview is given in Table 6.3 together with a reference to the applied libraries in the implementation. In case of basic PSO, no feature points are extracted. However, in order to reduce the amount of data, the orginal point cloud is downsampled using the *VoxelGrid* filter from PCL. The point cloud is put into a 3D voxel grid. For each box in the grid, all inlying points are approximated by their centroid. For each domain it is possible to select an individual error metric. The implemented metrics are listed in Table 6.4.

[8] `http://opencv.willowgarage.com/documentation/cpp/imgproc_feature_detection.html`

6.4.1 Class Hierarchy

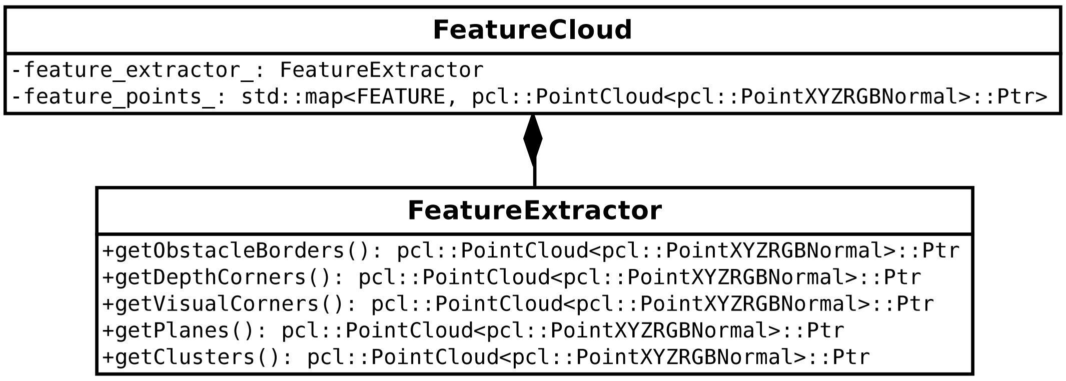

FeatureCloud
-feature_extractor_: FeatureExtractor
-feature_points_: std::map<FEATURE, pcl::PointCloud<pcl::PointXYZRGBNormal>::Ptr>
FeatureExtractor
+getObstacleBorders(): pcl::PointCloud<pcl::PointXYZRGBNormal>::Ptr
+getDepthCorners(): pcl::PointCloud<pcl::PointXYZRGBNormal>::Ptr
+getVisualCorners(): pcl::PointCloud<pcl::PointXYZRGBNormal>::Ptr
+getPlanes(): pcl::PointCloud<pcl::PointXYZRGBNormal>::Ptr
+getClusters(): pcl::PointCloud<pcl::PointXYZRGBNormal>::Ptr

Figure 6.12: Class hierarchy for feature clouds

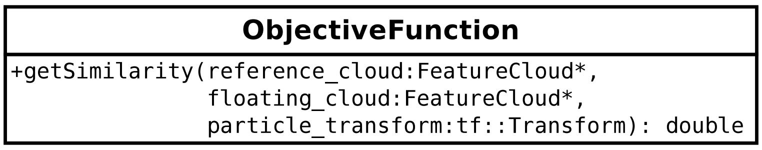

ObjectiveFunction
+getSimilarity(reference_cloud:FeatureCloud*,
floating_cloud:FeatureCloud*,
particle_transform:tf::Transform): double

Figure 6.13: Class hierarchy for similarity estimation

In feature-based PSO, point clouds are organized as a *FeatureCloud*. The class hierarchy of feature representations for reference and floating cloud is shown in Figure 6.12 and 6.13. An instance of *FeatureCloud* contains the original cloud and its normals. Normals are computed using *IntegralImageNormalEstimation* from PCL. There are several normal estimation modes available: COVARIANCE_MATRIX, AVERAGE_3D_GRADIENT and AVERAGE_DEPTH_CHANGE. In order to compute the normal for a point, 9, 6 or one integral images are created, respectively. In the implementation, the AVERAGE_3D_GRADIENT has shown to provide good results. The *FeatureExtractor* class is applied to extract feature points from the original cloud. They are stored as a map of feature points for each feature domain available in the *FeatureExtractor* class. Using these feature clouds, the *ObjectiveFunction* class computes the similarity distance between two feature clouds using the selected similarity measure and transformation defined by the particle pose (see Figure 6.13).

6.4.2 Visualization

A tool for testing of similarity measures was developed using *interactive markers*[9] in *rviz*. Interactive markers are geometrical primitives, which can be manipulated by the user in the 3D view of rviz. In this thesis, the transformation of such an item is accessed and applied to an "artifical particle" or "interactive particle". This means that the particle is not controlled by the swarm's behavior, but by the interactive particle's transformation. The interactive particle's position is controlled by the user in 6 DOF space. Thus, various distance measures can be applied to a reference and floating point cloud. The resulting cost values are visualized during the manipulation of the interactive particle. An interactive particle is visualized

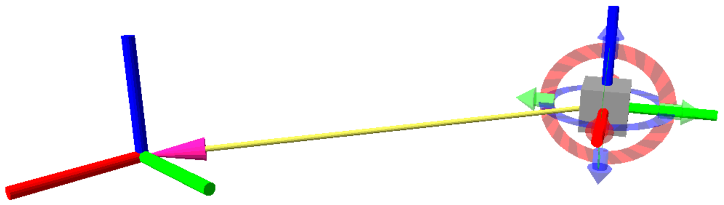


Figure 6.14: Camera coordinate frame (left) and interactive particle (right)

in Figure 6.14. The reference coordinate tree of the last known camera position is visualized on the left. The transformation vector is printed as an arrow from the interactive particle to the camera coordinate system. The marker has three rotational and translational axes. In Figure 6.15 and 6.17 the application of an interactive particle is shown. The reference point cloud is fixed to the camera coordinate frame and the floating cloud is transformed by the user. The particle has a context menu, which allows the user to reset the particle's position to the origin of the camera coordinate sytem, to get a new point cloud or add the current particle position as the new camera pose. In this example, the floating cloud is downsampled by a voxel grid filter and point-to-point distance is applied.

6.5 Summary

In this chapter, the implementation of a framework for PSO on RGB-D data was presented. An example for registered point clouds is shown in Figure 6.16. Initially, the connection of hardware components, a calibration procedure and logging of data for playback was introduced. A local pose estimation system, a library for PSO and tools for visualization and debugging were implemented. In the following Chapter 7 the implementation will be evaluated.

[9]`http://www.ros.org/wiki/interactive_markers`

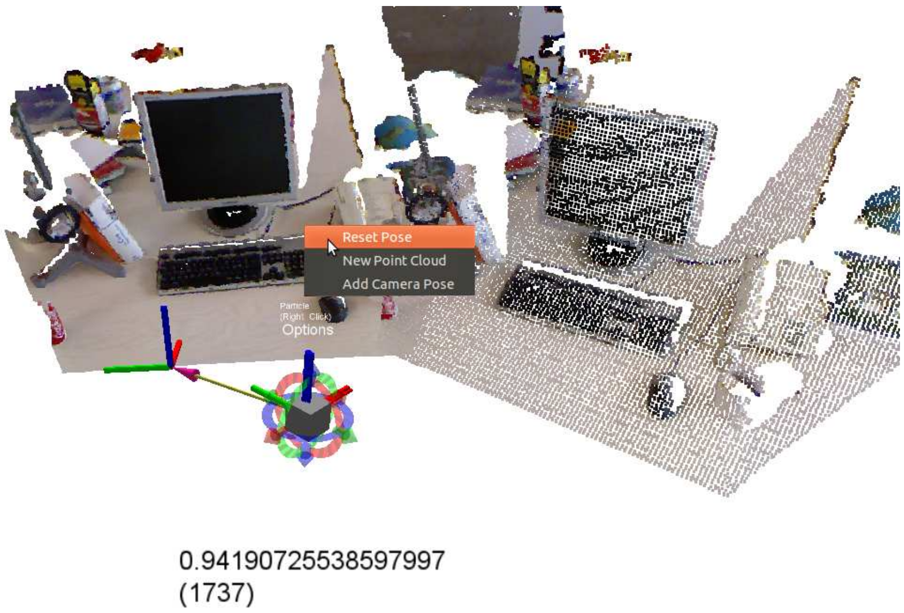


Figure 6.15: Application of an interactive particle with context menu and cost value of particle's transformation with the number of neighboring points in brackets on the bottom

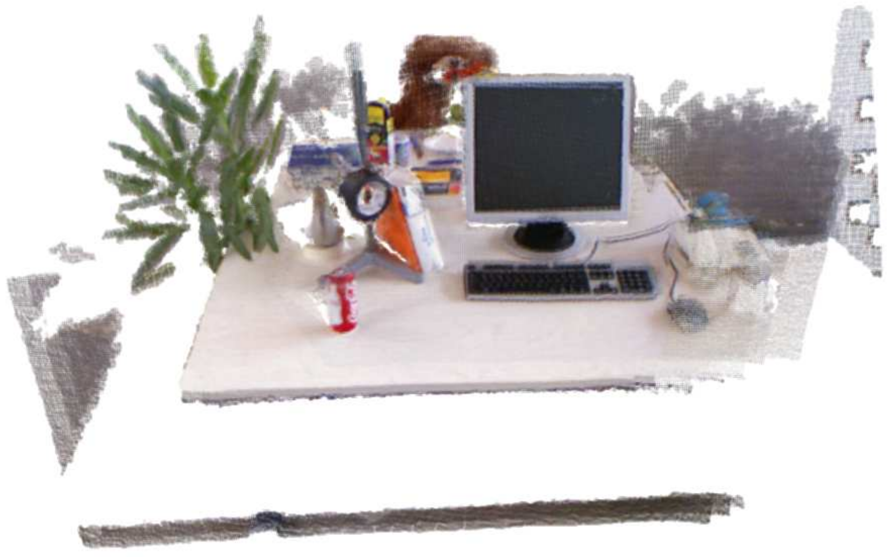


Figure 6.16: Registered point clouds using the PSO local pose estimation. The down-sampling step of the complete point cloud after each registration step leads to "layers" of points, which are visible on the outer scene areas caused by camera motion. The sensor data acquisition from different views has almost covered the invisible area behind the red can. Meanwhile, the points on the can itself are registered correctly.

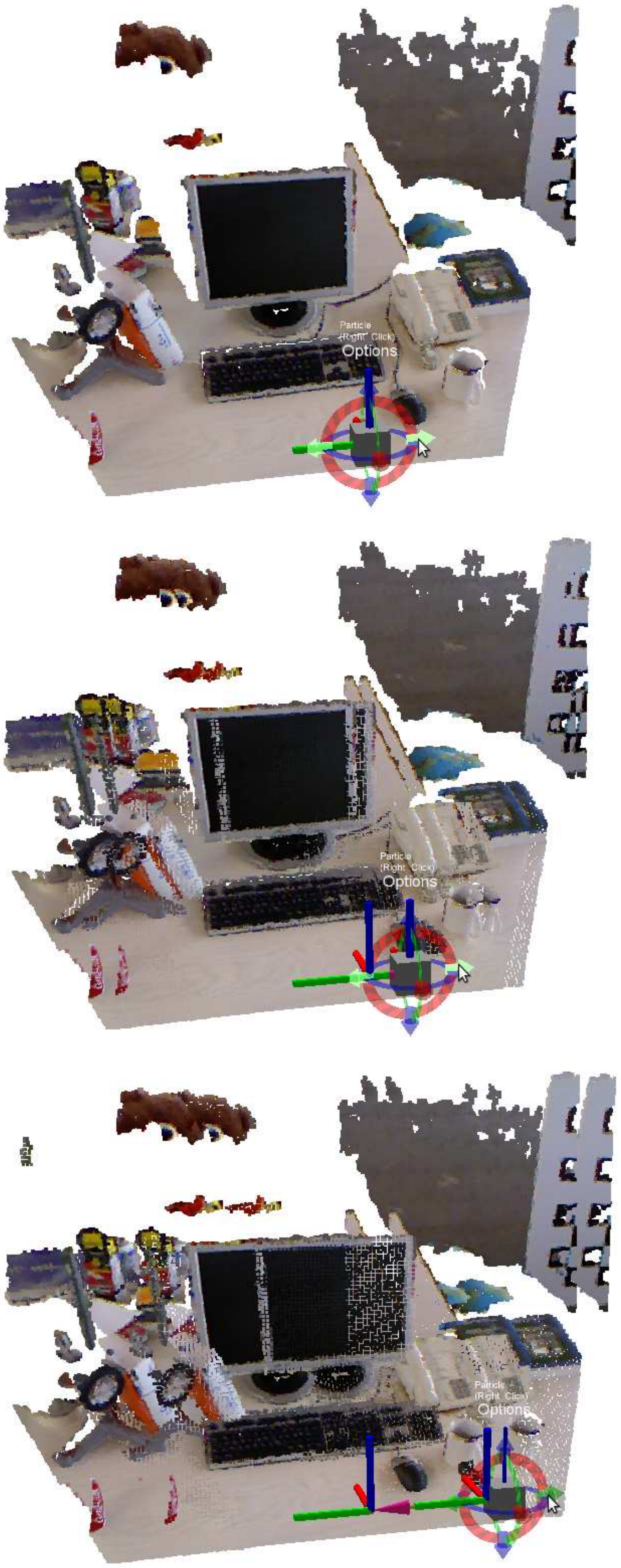


Figure 6.17: Application of interactive particle using point-to-point distance with cost 0 (28,683 correspondences), 0.0038 (28,683) and 0.1499 (24,733) from top to bottom

Chapter 7

Experiments and Evaluation

In this chapter an evaluation is performed, which compares the algorithms developed in this thesis to state of the art RGB-D SLAM systems. Furthermore, the configuration parameters of PSO are evaluated in order to optimize the configuration and understand the influence of parameters. In [KSD$^+$09] various approaches for the evaluation of SLAM system are presented. Feature-based approaches use the distance between measured and true landmark positions in order to evaluate the quality of algorithms. The evaluation of grid based approaches relies on visual inspection or overlaying with blueprints if available. However, comparing absolute positions has disadvantages as it may not be possible to overlay the data directly. In general, the comparision of map data originating from different algorithms and sensor data may not be possible due to different sensor modalities. In this thesis, the trajectory of the sensor is examined. A benchmark for the evaluation based on the sensor's trajectory is proposed in [SME$^+$11], which also provides a large database containing RGB-D data and ground-truth camera trajectories.

7.1 Database

The evaluation database of [SME$^+$11] contains the following data

- color and depth images of a Microsoft Kinect sensor:
  full frame rate of 30 Hz and sensor resolution of 640x480 pixels

- groundtruth trajectory of camera poses
  rate of 100 Hz

- IMU data from the accelerometer
  rate of 500 Hz

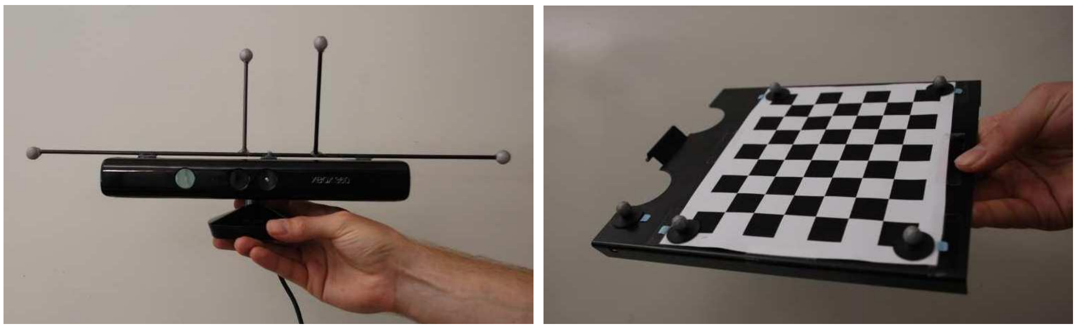


Figure 7.1: Experimental setup for camera pose recording in [SME+11]

Ground-truth data was generated by using a high-accuracy motion-capture system from MotionAnalysis with 8 high-speed tracking cameras at 100 Hz. Reflective markers were added to the Kinect as well as to a calibration board in order to track the sensor and the motion-capturing coordiante system (see Figure 7.1). The Kinect data was recorded using the PrimeSense OpenNi-driver with a Laptop running Ubuntu 10.10 and ROS Diamondback. The recorded scenes take place in typical office environments at various scales (one or more pieces of furniture up to a whole room) and contain trajectories with different translational and angular velocities. Furthermore, basic trajectories were provided for debugging, such as moving along or rotating around a specific axis. The data is stored as ROS bag files and available online[1]. The trajectories are stored in a file as 'timestamp tx ty tz qx qy qz qw' for every pose pair, which will be used for comparison with ground-truth data.

7.2 Quality Criterion

In order to compare trajectories, which were generated by a visual SLAM algorithm, an evaluation criterion is required. The following evaluation modes are available in order to compare the estimated trajectory with ground-truth data

- absolute trajectory error (ATE)
- relative pose error (RPE) for pose pairs

ATE is adequate for the evaluation of visual SLAM systems, because they aim at building a globally consistent model. In ATE, the difference of absolute pose values corresponding to timestamps is measured. The alignment of estimated and ground-truth trajectory is accomplished by singular value decomposition. The

[1]`http://cvpr.in.tum.de/data/datasets/rgbd-dataset`

difference between the resulting pose pairs is used to compute root mean square error, mean, median, standart deviation, minimum and maximum error. For visual odometry systems or local pose estimation as in this thesis, RPE is applied. In RPE the relative error between pairs of timestamps is computed.

7.3 Experiments

All experiments were executed using a Laptop (Intel Core i5 at 2,50 GHz with 4 GB DDR3 RAM) running Ubuntu 10.10 with ROS electric.

7.3.1 Evaluation of own approach

In this section, the configuration parameters of the local pose estimation system, which was developed in this thesis, is evaluated. The following sequences of the evaluation database [SME$^+$11] will be applied:

- *freiburg2_xyz* with translations in x-,y- and z-direction (Figure 7.2 left)
- *freiburg2_rpy* with rotations of roll-pitch-yaw (Figure 7.2 right)

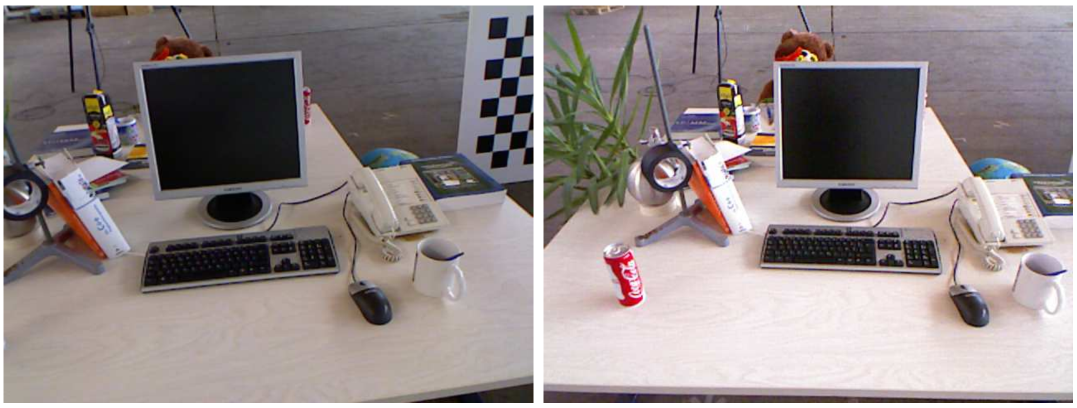


Figure 7.2: Sequences *freiburg2_xyz* (122.74 s) and *freiburg2_rpy* (109.97 s)

These sequences were recorded for debugging purposes: The camera motion is quite slow and there is almost no motion blur or shutter effects. Therefore, it fits perfectly for the evaluation of the parameters in this system. During the evaluation process the queue size is set to 1. Therefore, the number of registered pose pairs represents a measure for computation time.

Domains and similarity measures

In Table 7.1 point-to-point metric is evaluated for various domains. Borders and visual corners show similar error characteristics with borders allowing to estimate a larger number of pose pairs. Visual corners have lots of points in common with border points. However, unnecessary points are removed. Therefore, computation time is reduced and the camera movement to the next frame is smaller with faster registration. Planes show more accurate registration results than all other domains. The number of pose pairs for planes are rather small. The reason is that planes are not always available in a scene, cannot be extracted reliable or contain lots of points, which leads to higher computation time. The translational error for *freiburg2_rpy* is worse than for other domains, because planes per se cannot be located at a specific position. Depth corners have the highest number of registered pose pairs, but most results are worse than the other domains.

The evaluation of point-to-plane metric is listed in Table 7.2. For border points the results are better than in point-to-point distance measure. It also leads to a larger number of registered pose pairs. All other domains have higher error rates, which may result from an unstable normal estimation at these points. The

Point-to-point	translational (m)	rotational (deg)	pose pairs
Borders	0.0340 / 0.0402	1.5422 / 2.2693	93 / 64
Planes	0.0048 / 0.0581	0.0904 / 0.7692	36 / 16
Depth corners	0.0412 / 0.0498	1.6350 / 2.0117	135 / 47
Visual corners	0.0305 / 0.0448	1.3334 / 2.2765	62 / 47

Table 7.1: RPE mean error for point-to-point metric (freiburg2_xyz/_rpy)

Point-to-plane	translational (m)	rotational (deg)	pose pairs
Borders	0.0241 / 0.0428	1.0668 / 2.3500	107 / 98
Planes	0.0117 / -	0.2397 / -	40 / -
Depth corners	0.0676 / 0.0695	2.0408 / 3.5446	144 / 116
Visual corners	0.0332 / 0.0565	1.5675 / 2.9572	109 / 90

Table 7.2: RPE mean error for point-to-plane metric (freiburg2_xyz/_rpy)

Eucl. color	translational (m)	rotational (deg)	pose pairs
Borders	0.0334 / 0.0453	1.3515 / 2.5630	73 / 72
Planes	0.0055 / -	0.1298 / -	37 / -
Depth corners	0.0510 / 0.0616	1.8516 / 2.9820	129 / 85
Visual corners	0.0336 / 0.0375	1.3633 / 1.6442	62 / 46

Table 7.3: RPE mean error for euclidean distance of color (freiburg2_xyz/_rpy)

application of euclidean distance using color is evaluated in Table 7.3. Visual corners benefit from this type of distance measure. However, as in Table 7.1, planes are not registered successfully, because they do not have significant visual features. For HSV color distance, Table 7.4 shows similar results as euclidean distance of color, but with slight improvement. The sum of absolute distances similarity measure examines intensity values. It is faster to compute than HSV or euclidean distance, which results in a higher number of registered pose pairs (see Table 7.5). Even planes are registered successfully, but point-to-point distance has still smaller error values in this domain. Table 7.6 shows the pose error for NCC distance measure. Considering visual corners, it has smaller errors than SAD distance and less pose pairs. Comparing to other color or intensity distance measures, it has higher errors values in most areas.

HSV	translational (m)	rotational (deg)	pose pairs
Borders	0.0369 / 0.0458	1.5466 / 2.5201	78 / 73
Planes	0.0071 / -	0.2220 / -	34 / -
Depth corners	0.0485 / 0.0615	1.6759 / 2.8485	136 / 106
Visual corners	0.0256 / 0.0527	1.2439 / 2.4988	58 / 24

Table 7.4: RPE mean error for HSV color distance (freiburg2_xyz/_rpy)

SAD	translational (m)	rotational (deg)	pose pairs
Borders	0.0355 / 0.0589	1.6927 / 3.2225	118 / 103
Planes	0.0243 / 0.0193	1.1078 / 0.8599	38 / 20
Depth corners	0.0587 / 0.0888	2.3014 / 5.0014	171 / 130
Visual corners	0.0509 / 0.1112	2.1957 / 5.4905	111 / 96

Table 7.5: RPE mean error for sum of absolute distances of intensity values (freiburg2_xyz/_rpy)

NCC	translational (m)	rotational (deg)	pose pairs
Borders	0.0486 / 0.0577	2.1260 / 3.8492	92 / 89
Planes	0.0063 / -	0.1213 / -	21 / -
Depth corners	0.0657 / 0.1015	2.6672 / 5.6714	132 / 110
Visual corners	0.0616 / 0.0961	2.1098 / 4.3168	76 / 69

Table 7.6: RPE mean error for NCC distance (*freiburg2_xyz/_rpy*)

Summing up, the rotational error is always better when planes are considered (if they are available in the scene). The most successful similarity measure for planes is point-to-point distance. Borders always provide better results than depth corners, because they contain more robust points. The best results show optimization using a point-to-plane metric. If depth corners are examined, point-to-point metric is applicable. The distance of visual corners is best measured using euclidean distance of color.

Combination of domains

In order to benefit from each others characteristic, a combination of domains is evaluated in this section. Candidates are proposed in Table 7.7, which were derived from the previous evaluation results. Depth corners are omitted, because they have proven not to provide an advantage in accuracy towards border points.

Domain (metric)	translational (m)	rotational (deg)	pose pairs
Borders (point-to-plane)	0.0241 / 0.0428	1.0668 / 2.3500	107 / 98
Planes (point-to-point)	0.0048 / 0.0581	0.0904 / 0.7692	36 / 16
Visual corners (eucl. color)	0.0336 / 0.0375	1.3633 / 1.6442	62 / 46

Table 7.7: Candidates for a combination of domains (*freiburg2_xyz/_rpy*)

Domain (metric)	translational (m)	rotational (deg)	pose pairs
Borders (point-to-plane), Visual corners (eucl. color)	0.0030 / 0.0243	0.2022 / 1.1255	39 / 41

Table 7.8: Evaluation of borders and visual corners (*freiburg2_xyz/_rpy*)

Domain (metric)	translational (m)	rotational (deg)	pose pairs
Borders (point-to-plane)	0.0265 / 0.0406	1.2303 / 2.0987	188 / 196
Visual corners (eucl. color)	0.0400 / 0.0599	2.0104 / 3.7268	149 / 165
Borders (point-to-plane), Visual corners (eucl. color)	0.0274 / 0.0381	1.2295 / 2.0775	137 / 119

Table 7.9: Evaluation with long callback queue (length 100) (*freiburg2_xyz/_rpy*)

In Table 7.8 borders are combined with visual corners using point-to-plane and euclidean color distance, respectively. The combination of these domains shows an enhancement of error values for both translation and rotation. However, the combination of feature points leads to high computation cost. Therefore, less point pairs are registered compared to the application of single domains. The combination of borders and planes or visual corners and planes does not provide a reasonable number of pose pairs. The application of planes already results in a small number of registrations. If it is combined with another domain, the number of pose pairs decreases even more.

The evaluation shows that the quality of the registration suffers from high computation time, which leads to larger camera movements between frames and again to higher computation time to find a proper transformation. In order to measure accuracy with less influence of computation time, the queue size of the callback function is set to 100 instead of 1 in the evaluation of Table 7.9. The point clouds are received at a rate of 2 Hz. The experiment confirms that border points with point-to-plane metric provide the best result for computation time and accuracy. The overall number of registered pose pairs is higher compared to previous experiments, but the complete error is also increased. The reason for this behavior may result from an overflow of the queue, i.e. cached point clouds are omitted when more messages are received than processed. However, the high number of registered pose-pairs gives a realistic representation of the camera trajectory even if error rates are slightly higher.

7.3.2 Comparison to RGB-D SLAM

In this section datasets with faster camera movements than in the previous experiments are processed for evaluation. These sequences are shown in Figure 7.3. The local pose estimation system implemented in this thesis is evaluated against RGB-D SLAM by Engelhard et al. [Eng11]. The open-source implementation performs global registration on Kinect data using visual features as correspondences. SURF features are detected in the RGB images. Features are matched against reference features from the previous sensor measurement. 3D correspondences are determined from the feature correspondences using the depth image. The camera transformation is estimated by using RANSAC and further refined by generalized ICP. Finally, global optimization is performed by using a pose graph. However, for evaluation the global optimization step is omitted in order to compare it to the local pose estimation implemented in this thesis.

Table 7.10 shows the translational and rotational RPE for RGB-D SLAM. RGB-D SLAM reaches higher accuracy and a larger number of pose-pairs. SURF feature correspondences give a clue for the transformation between sensor measurements. The implementation in this thesis is not able to achieve equal or better registration results than RGB-D SLAM (see Table 7.11 for registration using PSO). However, other scenarios without significant visual features do not lead to successful registration. The approach in this thesis is more adaptable to different kind of features in the environment.

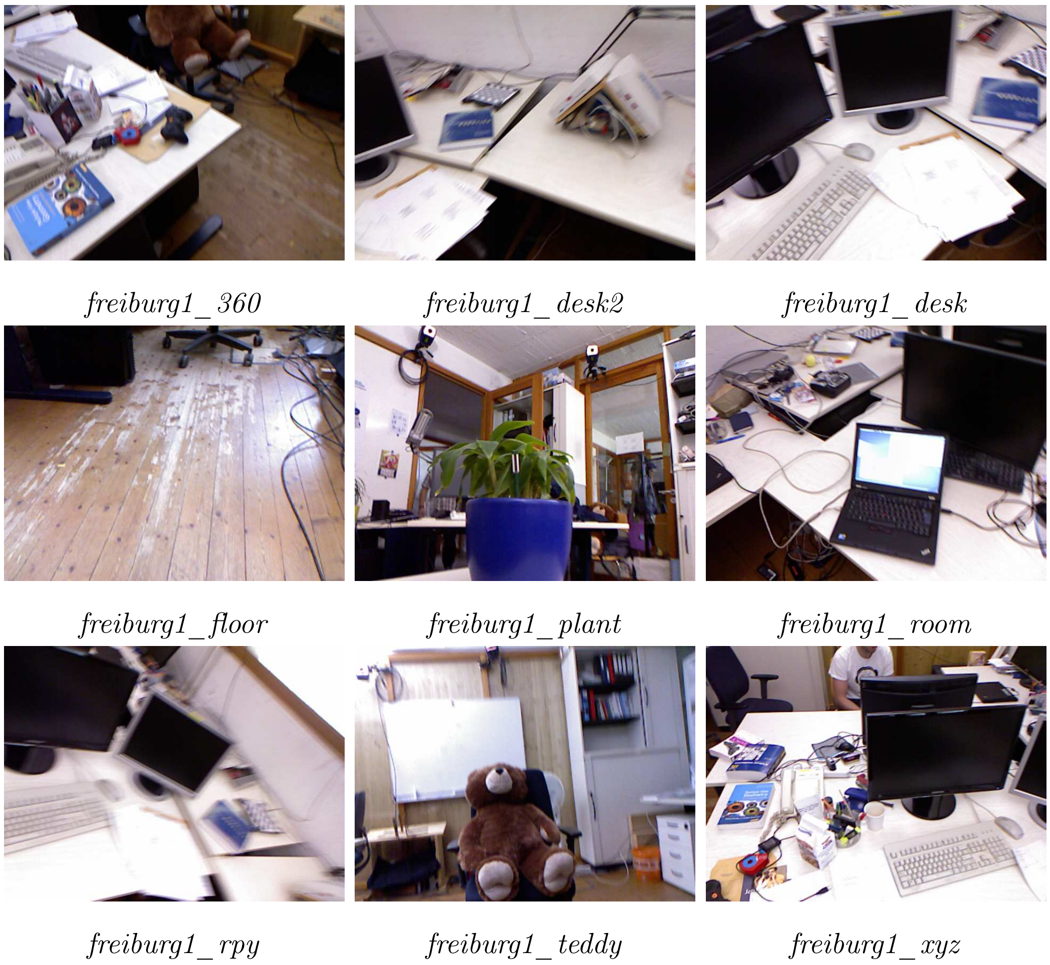


Figure 7.3: Sequences of evaluation database [SME+11]

Table 7.11 and 7.12 show the improvement of registration results, if border points and visual corners are combined. Both, translational and rotational error descrease significantly. The swarm contains 50 particles, which is a sufficient amount in order to converge in a global optimum. If too many particles are available, each iteration step becomes even more time consuming and does not allow the particles to explore the search space in various iteration steps. The normal distribution within the velocity update is set to $\sigma = 0.1$ even as the usually recommended value is $\sigma = 1$. The higher this value, the further particles jump away if they are likely to converge in a local optimum. In the meantime, they are not able to move through the search space accuratly if they jump too far away. Therefore a lower value provides better results as it still allows particles to jump in a sufficient range.

Sequence	translational (m)	rotational (deg)	pose pairs
freiburg1_360	0.0976	3.3670	496
freiburg1_desk2	0.0609	3.4712	574
freiburg1_desk	0.0483	2.4897	542
freiburg1_floor	0.0152	0.7476	1045
freiburg1_plant	0.1123	7.7840	1071
freiburg1_room	0.0656	2.8124	1150
freiburg1_rpy	0.0482	2.6326	645
freiburg1_teddy	0.1521	6.7401	1127
freiburg1_xyz	0.0255	1.2753	753

Table 7.10: RPE mean error of RGB-D SLAM without global optimization

Sequence	translational (m)	rotational (deg)	pose pairs
freiburg1_360	0.2597	15.0692	196
freiburg1_desk2	0.5384	23.6295	252
freiburg1_desk	0.3603	19.4626	173
freiburg1_floor	0.3796	20.6614	405
freiburg1_plant	0.3902	12.2048	288
freiburg1_room	0.3997	14.2119	546
freiburg1_rpy	0.1519	12.6398	190
freiburg1_teddy	0.3286	11.9628	459
freiburg1_xyz	0.2714	14.2810	296

Table 7.11: RPE mean error of PSO registration: border points (point-to-plane)

Sequence	translational (m)	rotational (deg)	pose pairs
freiburg1_360	0.2050	9.4347	145
freiburg1_desk2	0.3718	15.8563	127
freiburg1_desk	0.3970	15.3534	124
freiburg1_floor	0.3097	10.9196	125
freiburg1_plant	0.3818	11.0306	216
freiburg1_room	0.3627	12.8764	439
freiburg1_rpy	0.1321	9.4812	124
freiburg1_teddy	0.3287	9.2768	366
freiburg1_xyz	0.1972	10.7017	184

Table 7.12: RPE mean error of PSO registration: border points (point-to-plane) and visual corners (eucl. distance color)

7.4 Results

The experiments in this chapter shows that the local pose estimation system of this thesis is a good solution for the registration problem. One has to keep in mind that experiments using a particle swarm technique provide slightly different results for each test run, because of the random factor in the movement behavior of the swarm. In order to provide stable quality measurements, the experiments would have to be carried out for an infinite number of times repeatedly and averaged over time.

In the first part of the experiments, domains and distance metrics were evaluated considering accuracy and the number of registration pairs. Configurations for the comparison of this implemention towards other approaches were developed. RGB-D SLAM was evaluated against the new PSO approach with 9 different sequences of the database containing ground-truth information.

In the current configuration, the registration process is able to process 1 frame in 1-2 seconds when border points with a point-to-plane distance metric are applied. The optimization process rather struggles with the effects of high computation time than with accuracy. A larger movement distance between frames makes the optimization process even harder. An increase of framerate would be possible by altering parameters in order to optimize the algorithm for these sequences, e.g. a smaller number of particles, lower thresholds for convergence or to skip a cloud. However, these alterations form a tradeoff between performance and error rates.

The combination of domains and distance measures is able to improve the accuracy of registration results. Accuracy can be enhanced by using more feature points for registration or add more particles to the swarm. However, computation time remains a problem even if dense point clouds are reduced to feature clouds. Larger camera movements can be supported by a larger search space if more particles were available. In the current implementation the registration result is rather a question of computation time than accuracy.

Chapter 8

Conclusions

The increasing number of affordable color and depth sensors has made RGB-D registration a topic of high interest to a wide community. Not only robots need consistent 3D models of the environment for enhanced perception and understanding of the world, but also humans benefit from fastly captured and accurate 3D models. The algorithms presented in this thesis are able to create detailed environment models from continuous RGB-D sensor data. These models are the basic source of information for autonomous robots interacting with their environment. Partial views from only one sensor measurement are not sufficient to sense the complete environment. Tasks like 3D collision avoidance, object recognition and manipulation are based on perception. Also, further extraction of semantic information of the environment needs a reliable and accurate model of the world. By the establishment of these techniques mobile personal robots will be a daily source of support for the aging population e.g. by completing tasks like search-and-bring, pick-and-place or clean up.

8.1 Summary and Conclusions

One of the main challenges of local pose estimation of RGB-D data using a MPSO technique is optimization in high dimensional 6 DOF space. The search space has to be explored in order to let particles converge in a global optimum. Also, noise and the amount of data from the sensor are challenging for computation time and require efficient solutions. Finally, computational challenges arise from these difficulties. Approaches and solutions were discussed in order to solve these problems. These are the limitations of search space due to its application for registration of successive sensor measurements, parallelization of iteration steps during optimization, the usage of RGB and depth data and its reduction to feature points.

State of the art algorithms were introduced and divided in correspondence-based and search-space-based approaches. Commonly, local pose estimation is solved by correspondence-based approaches. In this thesis, a search-space-based approach is implemented. Its advantage is its ability to be used for global optimization, no need for a coarse registration step and the requirement of a simple distance measure, which projects reference and floating data to one distance value. Therefore, the need for various iteration steps until convergence should be neutralized. Techniques and applications of PSO were introduced and similarity measurement techniques for RGB-D data discussed. Both research areas are combined by using similarity measurement techniques as a cost function for MPSO. The cost values are applied to particles in order to find the correct local pose transformation of the camera in 6 DOF space.

A library for optimization using MPSO as in [KN10] was proposed. The modification in MPSO compared to the basic PSO approach originates in the application of a random factor with normal distribution to the movement of particles in search space. It allows the swarm to jump out of local optima compared to the implementation using only a uniform distribution. A tradeoff between exploration and exploitation of search space is given with the type of distribution of the random factor. The default values are zero mean and unit variance for this function. However, the larger the variance, the further away particles randomly jump. The exploration in search space is increased, but it is also likely that the swarm will miss the optimum. If the variance is rather small, the exploitation increases, but it is likely that the registration of larger camera movements fail. The default configuration of the variance was modified in order to enhance accuracy. In the following section on future work, an outlook will be given on how a variable variance could provide better results for registration.

The topology of the swarm defines the communication behavior of particles with one another. The influence of the social network is defined by the balance between social and cognitive component of the swarm. In this thesis, global, ring and wheel topologies were implemented. Ring topology was chosen as the most promising configuration for the swarm's network. Global topology converges fast, but often results in a local optimum. In this case all particles know the best global position of the swarm and are constrained in exploring the search space for other solutions. Wheel has a contrary behavior: each particle has only one partner in the network and therefore information is spread slowly. The ring topology, with two neighbors for each particle, has a good exploration and exploitation characteristic in search space. The dimensions of search space and other parameters of the MPSO library are configured by parameters. Therefore, the algorithm is easily adaptable to other problems which require optimization. Only the development of a function for the specific problem to be solved is required for cost computation in order to run optimization using the MPSO library.

For similarity estimation of RGB-D data a technique was developed, which extracts robust feature points from various domains such as planes, borders or corners from depth and color data. Various distance metrics were introduced to compute a cost value between two sets of feature clouds. The implementation is based on ROS and PCL, which are state of the art frameworks for robotic applications and point cloud processing. A visualization using the ROS visualization tool *rviz* was implemented, which allows the user to understand the internal behavior of the swarm by observing its movement.

In order to evaluate the output of the registration algorithm with a configuration of distance metrics and domains of feature types visually, a test mode was developed. It allows the user to control a particle by moving its position in search space. The particle determines the transformation of floating data and prints its cost function.

Aside from visual inspectation of generated 3D models during the registration of point cloud data, an evaluation was performed. So far, search-space-based algorithms per se were considered as slow when it comes to high resolution data. They are seldom used for 3D registration. However, the newly developed algorithms based on particle swarm optimization show that it is possible to solve this highly dimensional problem in adequate time. In this thesis, MPSO was executed on RGB-D data by reducing point clouds to distinctive feature points and contraining the search space. The iteration steps of MPSO were parallized so that the computation time for all swarm members is accelarated. This this way registration with an adequate camera movement distance is possible.

Constrains in computation time when using a PSO technique for point cloud registration still remain unsettled. Accuracy and the number of registered pose pairs were evaluated and compared to a state of the art correspondence-based search approach. The performance of PSO registration is about 1 frame in 1-2 seconds using border points with a point-to-plane distance metric. The optimization process rather struggles with the effects of high computation time than with accuracy. The number of feature points, which are determined by a selection of domains, are a tradeoff between performance and error rates. The swarm does not need a lot of members in order to converge in a global optimum. Already a small number of particles, e.g. 50 swarm members as in the experiments, are able to reach an optimum due to their smart swarm behavior.

The combination of domains and distance measures is able to improve the accuracy of registration results. Therefore, by putting more effort in optimization of computation time, local registration by MPSO is a competitive technique compared to other well known correspondence-based methods. A coarse registration step is not necessary and it is adaptable to different domains and distance measures. Therefore, further research in this area is recommended. This new approach has the ability to achieve competitive results by the implementation of the proposals in following section on future work.

8.2 Future work

The local pose estimation algorithm developed in this thesis has shown to provide good results. However, for a globally consistent 3D model, a global optimization technique is required. The small error between each frame-to-frame registration step increases to a larger error over time. A frame-to-model matching provides more consistent information.

The visualization of the generated 3D model from the estimated camera trajectory uses point clouds as a representation technique. Other methods, such as surface or volumetric data structures, are means to enhance models iteratively with more details and reduce the continuously growing amount of data. In this thesis, the generated 3D model is represented as a point cloud and resampled after each registration step by applying a voxel grid filter. The challenge of increasing data and request for accuracy is also a field of interest for building of semantic models of the world as 3D representations.

In 6 DOF space, the computation time for 3D registration using search-space-based approaches has always been a challenging problem. An idea to improve performance of the optimization is to enhance the initial distribution of particles in search space. The Kinect sensor is equipped with accelerometers and a tilt motor. In robotics, an Inertial Measurement Unit (IMU) is used to balance sensors on moving plattforms e.g. a 2D laser range finder mounted on a pan-tilt unit for map building. Here, the idea is to use the IMU data to enhance the initial distribution of particles in search space. If the sensor is pointing in a certain direction, the initial range of the particles' position and orientation should be set to this view. If the initial pose is a proper position, the optimization process will be able to converge faster and large errors in pose estimation will be avoided. The distribution of the swarm members could be adapted to the IMU data by implementing a changing variance of the random factor in the particles' movement.

Another promising approach to increase performance is a GPU based implementation. The parallel processing of large data benefits from GPU based implementations like it was already shown in the KinectFusion project. In this thesis, the accuracy of the estimated camera transformation is highly dependend on the quality of the detected feature points. The processing of complete point cloud data for similarity estimation in PSO would be interesting to evaluate considering performance and accuracy. The current CPU based implementation would not be able to live up to a reasonable performance in this configuration.

When it comes to accuracy, an interesting area of research is the determination of distinctive types of feature points and similarity measurement techniques. A comparison between the registration of feature points and dense data would be interesting considering its accuracy, because dense data actually provides more information but also contains more noise. In computer vision, 2D features are well known. In 3D the research for feature detectors and descriptors is still growing. Information like scale, color, normals and curvature are distinctive properties of point clouds, which were not available in this extent in 2D computer vision.

Appendix A

Algorithms

Listing A.1: Basic PSO due to [NdMM06]

```
randomly distribute $n$ particles in the search space
repeat
  for each particle i=1,...,$n$ do
    % set the personal best position
    if $f(\boldsymbol{x}_i) < f(\boldsymbol{y}_i)$ then
      $\boldsymbol{y}_i = \boldsymbol{x}_i$;
    end if
    % set the neighbourhood best position (global topology)
    if $f(\boldsymbol{y}_i) < f(\bar{\boldsymbol{y}}_{\boldsymbol{i}})$ then
        $\bar{\boldsymbol{y}}_{\boldsymbol{i}} = \boldsymbol{y}_i$
    end if
  end for
  for each particle i=1,...,n do
      update position using Equation 4.2 and 4.1
  end for
until stopping condition is true
```

Listing A.2: Modified PSO as Minimization Problem due to [KN10]

```
uniformly distribute $n$ particles in the search space
repeat
  **for** each particle i=1,...,$n$ **do**
    % set the personal best position
    **if** $f(\boldsymbol{x}_i) < f(\boldsymbol{y}_i)$ then
      $\boldsymbol{y}_i = \boldsymbol{x}_i$;
    end **if**
    % set the neighbourhood best position
    **if** $f(\boldsymbol{y}_i) < f(\bar{\boldsymbol{y}}_{\boldsymbol{i}})$ then
        $\bar{\boldsymbol{y}}_{\boldsymbol{i}} = \boldsymbol{y}_i$
    end **if**
  end **for**
  **for** each particle i=1,...,n **do**
    $\boldsymbol{p}^i = \boldsymbol{y}_i - \boldsymbol{x}_i$;
    $\boldsymbol{n}^i = \bar{\boldsymbol{y}}_{\boldsymbol{i}} - \boldsymbol{x}_i$;
    **if** sign($\boldsymbol{p}^i$ is same as sign $(\boldsymbol{n}^i)$ in all dimensions
      then
        update velocity $\boldsymbol{v}^i_{k+1}$ using Equation 4.3 with $N(0,1)$
      **else**
        update velocity $\boldsymbol{v}^i_{k+1}$ using Equation 4.3 with $U(0,1)$
      end **if**
      update position using Equation 4.1
  end **for**
until stopping condition is true
```

Appendix B

Mathematical Symbols

Glossary

P point cloud. 19

β constriction coefficient. 39, 41, 42

ω inertia weight. 36, 38–40

$\boldsymbol{c}_i$ cognitive component of particle i. 36, 37, 39–41

$\boldsymbol{p}_i$ point in three-dimensional space. 19

$\boldsymbol{r_1}$ random vector in cognitive component $[0, 1]$. 36–40

$\boldsymbol{r_2}$ random vector in social component $[0, 1]$. 36–40

$\boldsymbol{s}_i$ social component of particle i. 36, 37, 39–41

f definition of an objective function. 19, 51

j index of dimension in search space. 35, 36, 39, 41

k number of particles in neighborhood. 33, 34

n total number of particles. 33, 60, 90, 91

$\boldsymbol{\bar{y}_i}$ local best position of particle i. 35, 67, 90, 91

$\boldsymbol{v}_i$ current velocity of particle i. 35

$\boldsymbol{x}_i$ current position of particle i. 35, 67, 90, 91

$\boldsymbol{y}_i$ personal best position of particle i. 35, 67, 68, 90, 91

$f(\boldsymbol{\bar{y}_i})$ cost of local best position of particle i. 35, 67, 68, 90, 91

$f(\boldsymbol{x}_i)$ cost of current position of particle i. 34, 35, 67, 90, 91

$f(\boldsymbol{y}_i)$ cost of personal best position of particle i. 35, 67, 90, 91

Bibliography

[BGRM07] M. Ballesta, A. Gil, O. Reinoso, and O.M. Mozos. Evaluation of interest point detectors for visual SLAM. *International Journal of Factory Automation, Robotics and Soft Computing. v4*, pages 86–95, 2007. 44, 46

[BH01] K. Briechle and U.D. Hanebeck. Template matching using fast normalized cross correlation. In *Proceedings of SPIE*, volume 4387, pages 95–102. SPIE, 2001. 53

[Cle02] Maurice Clerc. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 2002. 32, 39, 42

[Eng11] Nikolas Engelhard. Real-time 3D visual SLAM with a hand-held RGB-D camera. *Pattern Recognition*, 2011. 27, 79

[ES04] Russel C. Eberhart and Yuhui Shi. Guest editorial special issue on particle swarm optimization. *IEEE Transactions Evolutionary Computation*, 2004. 31

[FK11] Nicola Fioraio and Kurt Konolige. Realtime visual and point cloud SLAM. In *Proc. of the RGB-D Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systems Conf.(RSS)*, 2011. 27

[GHT11] S. Gauglitz, Tobias Höllerer, and M. Turk. Evaluation of interest point detectors and feature descriptors for visual tracking. *International Journal of Computer Vision*, pages 1–26, 2011. 43

[GN09] Maik Günther and Volker Nissen. A comparison of neighbourhood topologies for staff scheduling with particle swarm optimisation. *KI 2009: Advances in Artificial Intelligence*, 2009. 32, 33

[Gos05] Ardeshir Goshtasby. *2-D and 3-D image registration for medical, remote sensing, and industrial applications*. Wiley-Interscience, 2005. 43, 44, 53, 54

[HHH98] R.L. Haupt, S.E. Haupt, and S.E. Haupt. *Practical genetic algorithms*. Wiley Online Library, 1998. 14

[HHRB11] Dirk Holz, S. Holzer, R.B. Rusu, and Sven Behnke. Real-time plane segmentation using RGB-D cameras. In *Proc. of the 15th RoboCup International Symposium*, 2011. 47

[HKH$^+$10] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. In *the 12th International Symposium on Experimental Robotics (ISER)*, 2010. 20, 22, 25, 27, 54, 55

[I H] I Heart Robotics. Limitations of the kinect (last accessed 23.11.2011). `http://www.iheartrobotics.com/2010/12/limitations-of-kinect.html`. 20

[IKH$^+$11] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, et al. Kinectfusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568. ACM, 2011. 20, 21, 28

[JCE06] Z. Jankó, D. Chetverikov, and A. Ekárt. Using a genetic algorithm to register an uncalibrated image pair to a 3D surface model. *Engineering Applications of Artificial Intelligence*, 19(3):269–276, 2006. 28

[JCE07] Z. Jankó, D. Chetverikov, and A. Ekárt. Using genetic algorithms in computer vision: Registering images to 3D surface model. *Acta Cybernetica*, 18(2):193–212, 2007. 28

[JKFB06] Patric Jensfelt, D. Kragic, John Folkesson, and M. Bjorkman. A framework for vision based bearing only 3D SLAM. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1944–1950. IEEE, 2006. 44

[KE95] James Kennedy and Russell Eberhart. Particle swarm optimization. *IEEE International Conference on Neural Networks*, 1995. 14, 31

[KN10] M. Khalid Khan and Ingela Nyström. A modified particle swarm optimization applied in image registration. In *2010 International Conference on Pattern Recognition*, 2010. 19, 32, 37, 38, 40, 42, 53, 64, 84, 91

[KSD$^+$09] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner. On measuring the accuracy of SLAM algorithms. *Autonomous Robots*, 27(4):387–407, 2009. 73

[LCE06] E. Lomonosov, D. Chetverikov, and A. Ekárt. Pre-registration of arbitrarily oriented 3D surfaces using a genetic algorithm. *Pattern Recognition Letters*, 27(11):1201–1208, 2006. 28

[LNY10] William Low, R. Nagarajan, and Sazali Yaacob. Visual based SLAM using modified PSO. *2010 6th International Colloquium on Signal Processing & Its Applications,*, 2010. 29

[MS11] F. Moosmann and C. Stiller. Velodyne SLAM. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 393–398. IEEE, 2011. 21

[NdMM06] N. Nedjah and L. de Macedo Mourelle. *Swarm intelligent systems*, volume 26. Springer, 2006. 14, 35, 38, 90

[Nüc09] Andreas Nüchter. *3D Robotic Mapping*. Springer Tracts in Advanced Robotics (STAR). Springer Verlag, 2009. 20

[Pri] PrimeSense. Primesense technology (last accessed 23.8.2011). `http://www.primesense.com`. 20

[QGC$^+$] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. ROS: an open-source robot operating system. 57

[RBB09] R.B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (FPFH) for 3D registration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3212–3217. IEEE, 2009. 25, 44, 47

[RCG11] R.B. Rusu, S. Cousins, and W. Garage. 3D is here: Point Cloud Library (PCL). In *Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China*, 2011. 19

[Rey87] C.W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH Computer Graphics*, volume 21, pages 25–34. ACM, 1987. 13

[Rus09] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Computer Science department, Technische Universität München, Germany, 10 2009. Advisor: Univ.-Prof. Michael Beetz (TUM) Ph.D.; Committee: Univ.-Prof. Dr. Nassir Navab (TUM), Univ.-Prof. Michael Beetz (TUM) Ph.D., Prof. Kurt Konolige (Stanford) Ph.D., Prof. Gary Bradski (Stanford) Ph.D.; summa cum laude. 19, 49, 51, 52, 68

[SBB05] L. Silva, O.R.P. Bellon, and K.L. Boyer. Precision range image registration using a robust surface interpenetration measure and enhanced genetic algorithms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(5):762–776, 2005. 25, 26, 28

[SC96] J. R. Smith and S.F. Chang. Tools and techniques for color image retrieval. *Storage & Retrieval for Image and Video Databases IV*, 2670:426–437, 1996. 54

[SC97] J.R. Smith and S.F. Chang. VisualSEEk: a fully automated content-based image query system. In *Proceedings of the fourth ACM international conference on Multimedia*, pages 87–98. ACM, 1997. 54

[SE98] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, 1998. 32

[SGB10] B. Steder, G. Grisetti, and W. Burgard. Robust place recognition for 3D range data based on point features. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1400–1405. IEEE, 2010. 46

[SHT09] A. Segal, D. Haehnel, and S. Thrun. Generalized-icp. In *Proc. of Robotics: Science and Systems (RSS)*, 2009. 25, 26, 27

[SME$^+$11] J. Sturm, S. Magnenat, N. Engelhard, F. Pomerleau, F. Colas, W. Burgard, D. Cremers, and R. Siegwart. Towards a benchmark for RGB-D SLAM evaluation. In *Proc. of the RGB-D Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systems Conf. (RSS)*, Los Angeles, USA, 6 2011. 22, 45, 47, 48, 49, 73, 74, 75, 80

[SMFF07] J. Salvi, C. Matabosch, D. Fofi, and J. Forest. A review of recent range image registration methods with accuracy evaluation. *Image and Vision Computing*, 25(5):578–596, 2007. 20

[SRKB10] B. Steder, R.B. Rusu, K. Konolige, and W. Burgard. NARF: 3D range image features for object recognition. In *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010. 44

[SRKB11] B. Steder, R.B. Rusu, K. Konolige, and W. Burgard. Point feature extraction on 3d range scans taking into account object boundaries. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2601–2608. IEEE, 2011. 46, 47

[WSZ$^+$04] M.P. Wachowiak, R. Smolíková, Y. Zheng, J.M. Zurada, and A.S. Elmaghraby. An approach to multimodal biomedical image registration utilizing particle swarm optimization. *Evolutionary Computation, IEEE Transactions on*, 8(3):289–301, 2004. 28, 32

[Zit03] Barbara Zitova. Image registration methods: a survey. *Image and Vision Computing*, 21(11):977–1000, 10 2003. 52