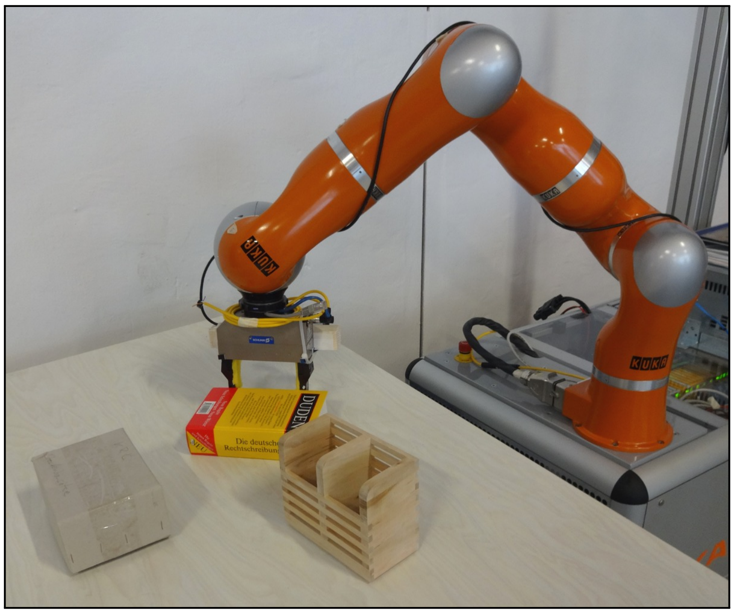
University of Freiburg
Faculty of Engineering
Department of Computer Science
Autonomous Intelligent Systems Group



Tactile Exploration for Segmentation of Objects

Master Thesis

Submitted By:	Daniel Kuhner
1st Examiner:	Prof. Dr. Wolfram Burgard
2nd Examiner:	Prof. Dr. Thomas Brox
Supervisors:	Christoph Sprunk, Jörg Röwekämper
Date:	February 25, 2013

Declaration

I hereby declare, that I am the sole author and composer of my thesis

Tactile Exploration for Segmentation of Objects

and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.

I hereby also declare, that my thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Daniel Kuhner

Freiburg im Breisgau, February 25, 2013

Zusammenfassung

Für sinnvolle Anwendungen im Bereich der autonomen Robotik sind Objektinformationen und -modelle der Gegenstände, mit denen interagiert werden soll, eine Grundvoraussetzung. Um die zeitaufwändige, manuelle Bereitstellung solcher Daten zu vermeiden, ist ihre autonome Generierung ein aktuelles Forschungsthema.

In dieser Arbeit wird ein Ansatz vorgestellt, der es einem Roboter ermöglicht die Objekte in seiner Umgebung durch autonome Interaktion zu segmentieren. Hierfür werden Objektkandidaten aus den Daten eines RGB-D Sensors extrahiert. Für die tatsächlich beweglichen Objekte wird dann ein 3D Modell generiert und segmentierte Farbbilder erstellt. Diese Daten können dann (nach Verarbeitung in weiteren Verfahren) die Grundlage sinnvoller Interaktion bilden.

Um zu prüfen, ob ein Objektkandidat beweglich ist, schlägt diese Arbeit die Interaktion mit dem Objekt durch Schieben vor. Die Interaktion mit dem Objekt wird außerdem zur Erzeugung von Änderungen in den Sensordaten verwendet. Diese Änderungen ermöglichen es, zu schlussfolgern, welche Teile der Sensordaten zum jeweiligen Objekt gehören. Durch wiederholte Planung und Interaktion wird das Objektmodel vervollständig. Wendet man diese Vorgehensweise auf alle Objektkandidaten an, erhält man für jedes bewegliche Objekt ein 3D Modell und die zugehörige Segmentierung.

Das vorgestellte Verfahren wird mit einem Roboterarm und einem RGB-D Sensor in einer echten Umgebung mit alltäglichen Objekten ausgewertet. Die Resultate zeigen, dass unser Verfahren in der Lage ist, akkurate Modelle von Objekten zu erzeugen. Des Weiteren zeichnet sich unsere Segmentierung durch eine hohe Genauigkeit aus. Unsere Experimente zeigen, dass eine zuverlässige Manipulation von Objekten durch Schieben möglich ist. Zusammenfassend erlaubt das in dieser Arbeit vorgestellte Verfahren eine autonome Erfassung von Objekten in der Umgebung als Grundlage sinnvoller Anwendungsszenarien.

Abstract

To perform any meaningful task autonomous robots need to be provided with information and models about the objects they should interact with. To overcome the tedious task of supplying these models by hand, autonomously acquiring such knowledge is an area of active research.

In this thesis we propose an approach that enables a robot to segment objects of interest from a working surface through autonomous interaction. The approach selects object candidates from a RGB-D sensor view. For the candidates found to be actual movable objects it generates a 3D model and segmented views of color images that can then be employed in further systems.

We propose to interact with object candidates through push actions to determine whether an object candidate is a movable object. These push actions are also used to generate changes in the sensor view that in turn serve to infer, which parts of the sensor view constitute the object of interest. By planning successive push actions on the current model, we iteratively improve the model of the object. Applying this method to all object candidates, we retrieve a 3D model and a segmentation of each movable object.

We evaluate our approach using a robotic manipulator and a RGB-D sensor in a real environment with everyday objects. Our results show that the proposed approach is capable of accurately modeling the objects. Furthermore, our segmentation is in high accordance to the ground truth. Our experiments show that the proposed pushing algorithm allows a robust manipulation of objects. Overall, our system enables an autonomous exploration of objects in the environment.

Contents

1. Introduction **1**

1.1. Contribution 4

1.2. Organization of the Thesis 4

1.3. Related Work 4

2. Overview and Background **11**

2.1. Object Modeling and Segmentation by Pushing 11

2.2. Notation 13

2.3. Mathematical Background 15

3. Object Modeling using Changes in Point Clouds **17**

3.1. Change Sets 17

3.2. Object Tracking and Modeling 22

3.3. Preprocessing of Point Cloud Data 23

3.4. Image Segmentation 25

4. Pushing Objects **29**

4.1. Using an Object Model for Manipulation 30

4.1.1. Object Configuration Space 30

4.1.2. Planning a Collision Free Path 34

4.2. Exploration of Unknown Objects 35

4.3. Object Manipulation 37
4.3.1. Physical Assumptions 37
4.3.2. Translational Manipulation 38
4.3.3. Rotational Manipulation 40

5. Evaluation **43**
5.1. Experimental Setup 43
5.1.1. KUKA omniRob with Light Weight Robot 44
5.1.2. Schunk WSG Gripper 44
5.1.3. ASUS XtionPRO LIVE 45
5.1.4. Software 45
5.2. Experiments 46
5.2.1. Quality of the Object Model 47
5.2.2. Segmentation Quality 49
5.2.3. Accuracy of the Manipulations 52
5.2.4. Table Plane Estimation 54
5.2.5. Filtering Point Clouds 56
5.2.6. Parameters 58

6. Conclusions **61**

A. Mathematical Background **65**
A.1. Eigenvectors and Eigenvalues 65
A.2. Singular Value Decomposition 66
A.3. Principal Component Analysis 67

B. Algorithmic Basics **69**
B.1. Normal Vector Estimation 69
B.2. RANSAC 70
B.3. Potential Fields 72
B.4. The A* Algorithm 73
B.5. Iterative Closest Point 74
B.6. Concave Hull Estimation 75

Bibliography **77**

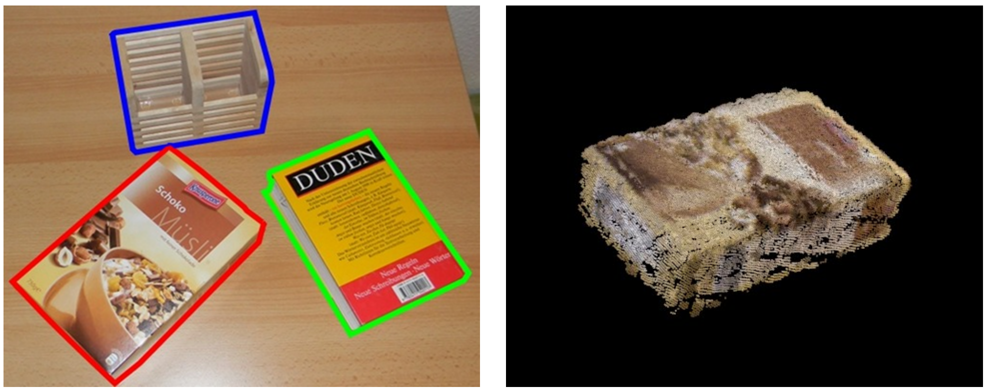
List of Figures **85**

1 Introduction

In the current state of the art in service robotics 3D models and segmented images play an important role. They are the key to enable autonomous interaction of robots with its environment. They are needed for, e.g., grasping or detecting objects. The object model helps to find suitable grasps, while the segmented images can be used to train an object classifier or detector. Often, manual segmentation is used to obtain such training data. The process is supervised, and thus time-consuming and therefore also expensive. The object model is also not readily available for many real-world objects. It must be generated manually using a CAD software or with a 3D scanner, e.g., RGB-D sensors or laser scanners.

Example applications, where object detection and manipulation plays a fundamental role, are found in the household service context. Think about a robot that needs to carry out household activities like setting up or cleaning a table or fetching items. Without knowledge of its environment, the robot has to be capable of learning how objects in the environment look like. Figure 1.1 shows a scene with different objects. An interaction task, e.g., tidying up the table, requires information about them. For carrying out household activities it is also important to decide whether an object can be moved or if it is rigidly connected to the environment. A simple manipulation is often enough to decide if they are movable. Furthermore, object detection is a basic skill for household robots, e.g., for searching an object that is to be fetched.

We present an approach to autonomously acquire such enabling informa-



(a) Scene with segmentation

(b) Reconstructed 3D model of the lower left object in Figure (a).

Figure 1.1.: For further processing of the objects, as grasping or object detection, it is advantageous if the objects are known in-depth. Our approach is designed to autonomously extract a three-dimensional object model and a segmentation of the color image for each object. Furthermore, since we manipulate objects, we can be sure that the objects are movable and not rigidly connected to the scene.

tion. The problem is divided into two components. First, we propose how to model objects and how to segment them from the environment, and second we present our approach for manipulating objects.

Our object modeling algorithm is based on motion of the objects. Such motion causes a change in the sensor readings that were taken from the scene. We denote these changes as *change set*. The changes belong to the object if the assumption holds that only one object is moved between two consecutive sensor readings of the scene. Using the changes we can compute the motion of the object, i.e., the transformation, which allows to combine several change sets into a single three-dimensional object model. In addition, if no changes were detected in the sensor data, the object has not moved. Thus, the corresponding object is assumed to be not movable and can be neglected in further processing. By iteratively performing the above steps, we get a more and more complete object model. Furthermore, we project the object model into the corresponding color image of the current view to retrieve the segmentation of the color image.

To allow an autonomous exploration of the objects in the environment an active manipulation is required. There are various ways for a robot to manipulate the scene at hand, where grasping is the most popular one. It has the advantage that it allows to lift up an object from the scene and transport it to another location, e.g., from a table to a book shelf. Furthermore, it is a good technique for reconstructing objects since they can be rotated by the manipulator in front of the sensor to scan the complete surface.

However, grasping is not applicable if the objects are too heavy or too large for the manipulator. It is also problematic if no object model is given to compute the grasping points. Another possible way for manipulation is pushing. Pushing has some advantages over grasping. It allows to manipulate larger and heavier objects. Additionally, the hardware that is required for pushing is typically cheaper and less complex compared to grasping hardware. For example, the manipulator can consist of a single finger, whereas grasping requires at least two movable fingers or some special hardware (e.g., vacuum based). Aside from this, pushing is mostly a two-dimensional planning problem, since pushing is done on a surface – the manipulation planning is thus less involved when compared to grasping, which is done in 3D.

Nevertheless, stable pushing is not an easy to solve problem. For a perfect motion prediction we need the support distribution (how friction acts in a location dependent way) between the object and the working surface. It is needed to compute the center of rotation. Its location is highly dependent on the support distribution. For instance, an object that has a buckling bottom surface has another center of rotation than an object with a flat surface. Unfortunately, support distributions are hardly available. This means that we cannot predict the motion perfectly – we need to rely on some approximating assumptions.

It becomes apparent that the geometric center is a sufficient approximation in our setup. Our approach describes how the manipulation is planned to avoid collisions with other objects. Furthermore, we suggest to execute translational and rotational push actions separately. This allows a robust interaction with the objects.

1.1. Contribution

In this thesis we propose an approach that enables a robot to autonomously segment objects of interest from a working surface through active interaction. In particular, we introduce

- a set of rules to infer which of the changes in RGB-D sensor readings caused by a displaced object belong to the object itself,
- an approach to merge a number of such changes for an object into a consistent 3D model and color image segmentation,
- a method to displace an object by a desired amount through a push interaction,
- and a coordinating framework that without prior knowledge detects object candidates on a working surface and subsequently plans and executes manipulations to incrementally build 3D models and segmentations for the actually movable objects.

1.2. Organization of the Thesis

The remainder of this thesis is organized as follows. After we discuss related work, Chapter 2 gives an overview of the proposed system. Then, in Chapter 3 we introduce the object modeling and segmentation. The object modeling is based on changes between point clouds. They are the result of an autonomous manipulation of the scene. The manipulation is done by pushing and is described in Chapter 4. Afterwards, Chapter 5 evaluates the proposed approach by a series of real-world experiments. Finally, Chapter 6 concludes the thesis.

1.3. Related Work

In the context of autonomous training data generation, i.e., object modeling by manipulation and object manipulation by pushing, very recently, a few papers were published. Hermans et al. [19] propose an approach that uses object manipulation by pushing for separation of object clusters. Unlike our approach, they extract visual edges to detect objects. The visual edges are used as an initial push location. ICP (Iterative Closest Point) [52] is then used to determine the motion of an object. Each object cluster is assigned to an orientation histogram containing n bins. Such a histogram keeps track of already executed pushing actions in the corresponding direction. As soon as every bin is nonzero, an object is assumed to be separated from others. If a cluster is split by pushing, two new clusters are generated with new orientation histograms. We also use ICP and changes in point clouds to estimate the motion after the manipulation. However, we are not only interested in separating objects. We also want to generate a three-dimensional model in an autonomous way. Therefore, the pushing actions must be planned because we need to get a view from each side of the object.

The segmentation of a pile of objects was recently done by Katz et al. [25]. A segmentation algorithm based on edge detection in the depth image and normal discontinuities provides an initial guess about objects. This guess is then verified by manipulation and a subsequent check if this segmented patch in the old scene can be detected in the scene again by a set of features, e.g., SIFT, color histograms, etc. Finally, reliably detected objects are removed from the scene. In this thesis the tracking step is done by ICP. Our aim is object modeling and segmentation for generating training data. Thus, we need to compute a transformation between different object views to merge them.

Another closely related approach was proposed by Krainin et al. [27]. They describe how a robot with a manipulator can be used to model a grasped object. By rotating the objects in front of a depth sensor a three-dimensional mesh is generated. The approach selects the next view position by sampling positions around the object. Each candidate is then evaluated considering the view quality and the actuation costs. Due to occlusions that are caused by the manipulator the robot has to put the object on the table and computes a new grasping position to map the whole object. For mapping they propose a probabilistic method based on signed-distance functions. Their approach is only applicable, if the object is graspable. To allow processing of non-graspable objects, we propose manipulation by pushing. We also do not need to consider filtering out the manipulator, since the manipulator can leave the scene before taking a sensor reading.

In the following, the related work in the area of object modeling, manipulation and segmentation is discussed.

3D Reconstruction Based on different object views we build a 3D object representation based on point clouds. Most of the related object modeling algorithms are based on the setup of a moving camera and a static scene. In our approach, the setup is the other way round. The camera is static and a single object is moved in the scene. In this thesis, the computation of the transformation between two object views is done by ICP. Each view is then integrated into a model that is represented by a point cloud.

A high fidelity approach for 3D reconstruction – named "KinectFusion" – was proposed by Izadi et al. [23]. The Kinect depth sensor from Microsoft is moved around to create a three-dimensional model of an indoor scene. The depth data is used to track the sensor position. The views are combined to retrieve an accurate object model. To allow a real-time computation, Kinect-Fusion is implemented on the GPU (graphical processing unit). It also allows augmented reality applications such as virtual interaction with the scene or physical particle simulations (see Figure 1.2).

A similar approach was published by Henry et al. [18]. They use RGB-D features (SIFT) to estimate correspondences between views. Then, RANSAC is used to filter out bad correspondences. Finally, ICP aligns the views using the transformation that is estimated from the visual features. Their approach is however not capable of modeling a single, segmented object as we do.

Object Manipulation The robotic manipulation of an object can be done in different ways. The most common is grasping. Rao et al. [37] propose an approach to grasp novel objects for cleaning a cluttered table. They segment the image by examining depth information. A supervised learning classifier decides whether a segment in the image is suitable for grasping. The pro-

Figure 1.2.: KinectFusion: A related approach for precise 3D reconstruction in real-time. It also allows an augmented particle simulation [23].

posed approach uses the three-dimensional point cloud to plan an antipodal grasping motion (i.e., the fingers of the manipulator are placed on opposite points of an object).

Saxena et al. [42] also present a method to grasp unknown objects. Opposed to the approach by Rao et al. they do not rely on a three-dimensional model of the object. Potential grasp points are located in two or more images of the object. A triangulation of these points obtains the 3D location for grasping. They use a supervised learning algorithm to identify grasp locations in the images.

Hsiao et al. [20] analyze how an object can be grasped to deal with uncertainty. They conclude that grasping human-designed objects is often possible by starting from above or from the side of the object. Then, the manipulator is aligned with the principal axis of the object. Finally, the manipulator tries to grasp the object around its center. In addition, tactile sensors can be used to compensate uncertainty while grasping.

One problem with grasping is that it cannot be applied to objects too large or heavy for the employed manipulator. Grasping also requires a quite good understanding of the object's shape to be accurate. However, if we only want to move objects on a planar surface, e.g., a table, pushing and pulling are alternatives to grasping.

Pulling has disadvantages in the hardware context. It requires to grasp the object or it needs some special hardware to maintain contact with the object during motion, e.g., using a rope that connects the object and the robot. On the other hand, pushing is feasible with less complex hardware. We argue to use pushing because we assume that objects are unknown and can be too heavy for lifting them or too large to be grasped by a gripper.

One of the first works in the context of robotic pushing was presented by Matthew Mason in the 1980s. He studied the physical basics of object pushing [30, 31]. Mason assumed that the motion of the object is *quasi-static*, i.e., there is always a contact point between the object and the manipulator – this can be achieved by slowly pushing. It turns out that the complete prediction of pushing is only possible if the support friction between object and environment surface is known.

Peshkin and Sanderson [35] explain how the center of rotation can be determined if the support friction is known. Lynch and Mason [29] show how to push objects without knowledge of the support friction if multiple contact points are available, e.g., when using a gripper with two fingers. They search for pushing directions, where the object remains fixed to the manipulator, namely the stable pushing directions. We approximate the center of rotation since the support distribution is unknown for many real-world objects.

Mataric et al. [32] use two communicating robots to accomplish box pushing. Both robots push cooperatively to retrieve more stable and faster results. The combination of pushing and grasping is advantageous if objects are located side by side in a setup with high sensor noise. Dogar and Srinivasa [12] present an approach to resolve situations in which direct grasping is not possible. The manipulator is used to push the object into free space until grasping is possible. A similar approach was presented by Omrcen et al. [34] (see Figure 1.3). Their publication describes an algorithm to learn how objects will react to manipulation. There, a robot executes many different pushing manipulations and observes the object after each manipulation. In doing this, their approach takes at least 50 steps to get adequate results. Using this learned model, an object that cannot be grasped directly can be pushed to another location for grasping.

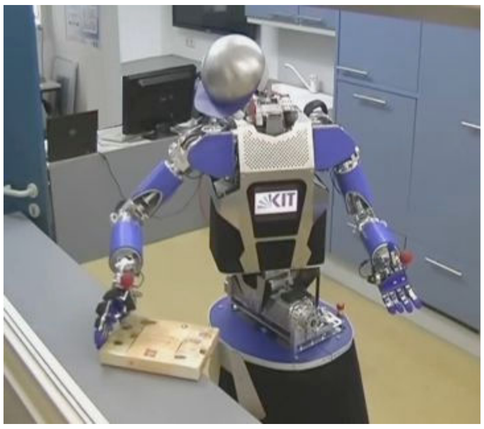


Figure 1.3.: Pushing an object to enable grasping: Since the manipulator is not able to grasp it directly, the robot pushes it to the edge of the table for grasping. The physical parameters are initially unknown and learned by a series of manipulations [34].

Segmentation Many approaches that were presented in the last years describe different techniques to segment images. Segmentation approaches can be based on variational methods. They use energy minimization to find the segmentation. Examples of such methods are built with level sets or graph cuts. Level sets are three-dimensional functions whose zero pass describes the segmentation of the two-dimensional image [28, 53]. Shi and Malik [45] propose a segmentation algorithm based on a graph partitioning problem. They use *normalized cuts*, which is a global measure for both the dissimilarity between different groups (differently labeled areas) and the similarity within a group.

There exist statistical approaches for segmentation, too. These can be regarded as estimating the most probable contour in the image [6, 26]. There are various other techniques to segment an image, including contour-based segmentation [4] or simple thresholding.

Segmenting an image is often ambiguous, since the segmentation can only use the data of one image, e.g., contours, colors and so on. In many cases, this results in a poor segmentation quality. When motion comes into play, i.e., in a video sequence, segmentation gets more accurate, since coherently moving areas in a sequence of images are more likely to belong to a single object. Thus, we also rely on motion in this thesis to model and segment objects.

In the field of motion segmentation, which is based on time dependent two dimensional data, Brox and Malik [9] propose an unsupervised approach using long-term point trajectories. Such point trajectories are obtained using an optical flow based tracker on a video sequence. The trajectories are then labeled by a clustering algorithm. Occlusions can also be handled. Due to computational reasons point trajectories are sparse. Using a hierarchical variational approach, the sparse segmentation can be turned into a dense one [33]. Based on optical flow many other approaches were proposed [10, 44, 48, 51].

By building a three-dimensional model from all changes, the segmentation of the color image can be done using a projection from the 3D space into the image space. While the above mentioned motion dependent approaches need to track points in the image, our approach uses ICP for matching three-dimensional point clouds from different views. Furthermore, our approach actively triggers the motion itself.

2 Overview and Background

First, we give an overview over all components of the proposed system and how they are combined. Subsequently, the notation, which is used throughout this thesis is introduced. Finally, we discuss the mathematical preliminaries and basic algorithms.

2.1. Object Modeling and Segmentation by Pushing

The key idea of the approach is to use a manipulator to examine an unknown scene by pushing objects. Therefore, we place a manipulator in front of the objects that have to be explored. A statically mounted RGB-D sensor provides a point cloud and color image of the scene. Based on this, the system decides, which parts of the point cloud are movable object candidates using a heuristic. The object candidate is then pushed by the manipulator to generate a motion. Before and after each push action, the RGB-D sensor provides new data. By analyzing two consecutive point clouds we can check if a motion took place. Motion causes a change of depth in the new point cloud, where depths are the distances from the points to the sensor. The changes are used to compute a change set, which contains points of the moved object. The rules for computation are later on described in details. Based on the information of the change set, the segmentation and modeling is done. If a motion took place, we know that the object candidate is movable and can

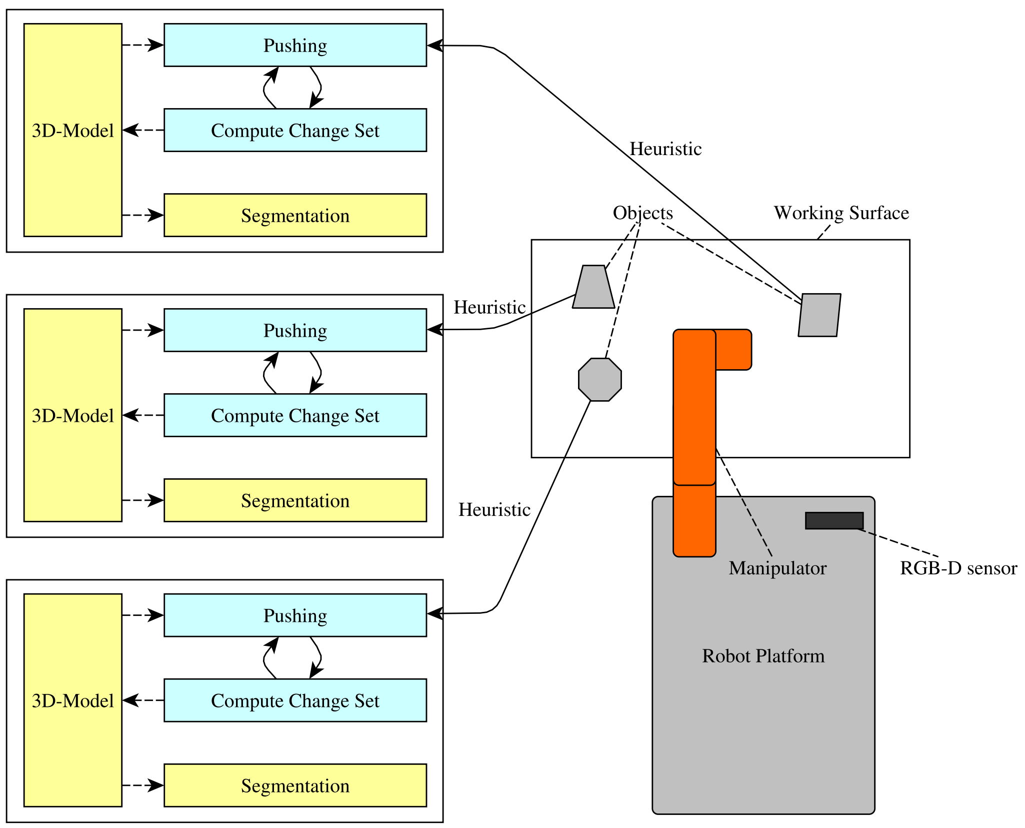


Figure 2.1.: System overview: A manipulator is used to push objects on a planar surface. By pushing and sensing the changes induced by a moving object we can segment it from the background and estimate a 3D object model.

be further explored. On the other hand, if the object did not move, we can conclude that it either belongs to the background, it is blocked by another object or it is too heavy. Furthermore, it is possible to combine the point cloud parts that have changed into a three-dimensional object model. Iteratively, the quality of the segmentation and object model can be improved by incorporating the results of new push operations. Once the object is regarded as fully known, the next movable object is considered. Figure 2.1 shows how the approach is designed.

We do not need to know the object's pose and 3D model before executing the first push operation. Nevertheless, we assume that

- the objects are located on a planar surface, e.g., a table,
- the objects that have to be segmented are reachable by the robot manipulator (other objects are ignored),
- the objects are rigid,
- only one object is pushed at once and
- the RGB-D sensor's point of view is constant over time (but it is not necessary to continuously acquire sensor data).

Whenever an object is moved, a change between consecutive point clouds can be detected by sensing the scene before and after every pushing action. This change has two consequences that we are interested in:

1. The object has moved, which implies that the object is not rigidly connected to the background, i.e., the working surface.
2. The change between two successive point clouds can be regarded as part of the pushed object. We explain this in more detail in Section 3.1.

In conclusion, the iterations of pushing and sensing result in a segmentation and a 3D model of the corresponding object. If an object is regarded as fully known, the next potential object will heuristically be determined from the unknown scene until all objects are known. Thus, after all objects have been processed, it is possible to segment the complete scene into background and reachable, movable objects.

2.2. Notation

We will write position vectors as bold lowercase (e.g., $\mathbf{x}$), while direction vectors are written with an arrow on top of the letter (e.g., $\vec{\mathbf{n}}$). Matrices are bold uppercase letters (e.g., $\mathbf{R}$). Sets will be denoted as script letters, e.g., $\mathcal{C}$.

We use the L$_2$-norm for vectors $\mathbf{x} \in \mathbb{R}^n$ denoted by

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}. \tag{2.1}$$

Furthermore, we use the L$_1$-norm (Manhattan distance) between $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^n$

$$\|\mathbf{x} - \mathbf{y}\|_1 = \sum_{i=1}^{n} |x_i - y_i| \tag{2.2}$$

for distance computation in a grid world.

In the following, we define the basic data we use in the remaining chapters. A point cloud is given as a set of n points $\mathbf{p}_i \in \mathbb{R}^3$

$$\mathcal{P} = \{\mathbf{p}_i \,|\, 0 \leq i \leq n\}. \tag{2.3}$$

A *change set* contains the points which appear or disappear between two views (see Section 3.1).

The normal of a point $\mathbf{p}_i$ in $\mathcal{P}$ is the vector

$$\mathbf{n}_i \in \mathbb{R}^3, \quad \|\mathbf{n}_i\|_2 = 1 \tag{2.4}$$

of length one. Its direction is orthogonal to the surface of $\mathcal{P}$ at $\mathbf{p}_i$. In practice, the normal is estimated on a surface patch around $\mathbf{p}_i$. Section B.1 explains, how the direction of the normal is determined.

Furthermore, the geometric center of a point cloud is estimated by the arithmetic mean over its points:

$$\mu = \frac{1}{|\mathcal{P}|} \sum_{\mathbf{p} \in \mathcal{P}} \mathbf{p}. \tag{2.5}$$

2.3. Mathematical Background

Before the approach is introduced, we give a short overview over the mathematical background and algorithms that we use in the thesis.

Principal component analysis (PCA) is a tool for converting a set of values into linearly uncorrelated values. The principle components are orthogonal and in the direction of the largest variance of the data. We use PCA to estimate the size of an object, and with it the distance an object must be moved during manipulation. The PCA can be solved using a *singular value decomposition*. It describes a decomposition of a matrix into three special matrices and is closely related to the *eigenvalue decomposition*.

We need to know the normals of point clouds to predict how an object must be manipulated to move it to a defined goal. Since manipulation by pushing is basically done on a surface (i.e., the object is not lifted up), we assume that the objects are placed on a planar surface. Thus, the planning and movement prediction requires the estimation of the working surface (e.g., the table). We use the *RANSAC* algorithm to estimate it.

Given the working surface a planning step is needed for manipulation. It is performed via the *A** algorithm in the configuration space of the object. The configuration space contains all possible object configurations (i.e., represented by its location $(x, y) \in \mathbb{R}^2$ and the rotation $\theta \in [0, 2\pi)$) on the working surface. Using A*, a collision free path can be planned to manipulate the objects.

Each configuration of the configuration space is assigned to a score that describes the quality of the corresponding configuration. We model the scores with *potential fields* consisting of *potential values*. Finally, the tracking of objects is done via the *iterative closest point* algorithm. It estimates a transformation to align two point clouds.

For a more detailed presentation of the algorithms and concepts mentioned above we refer the reader to Appendix A and B.

Summary

We use a robot to manipulate objects by pushing. A three-dimensional object model is determined based on the change in the point clouds caused by the manipulation. The object model also allows to estimate the two-dimensional segmentation of the color or depth image.

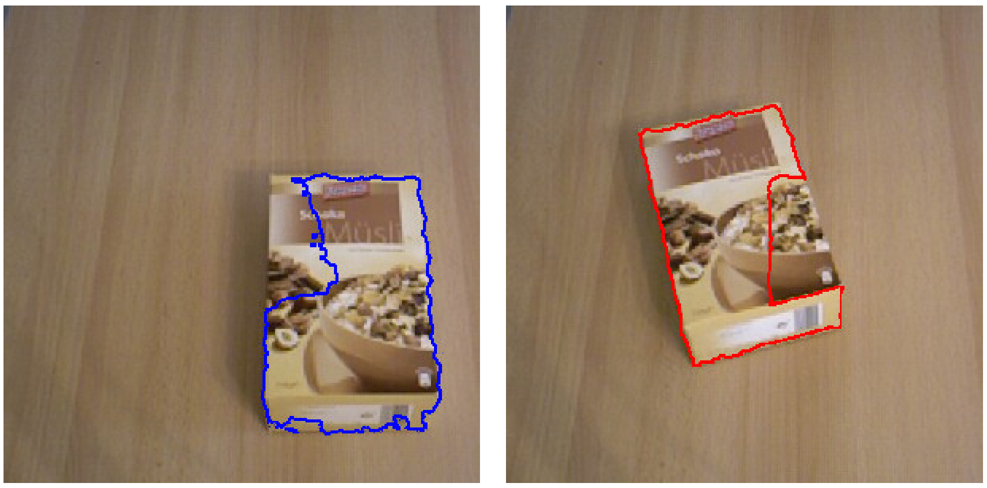
3 Object Modeling using Changes in Point Clouds

We base our object modeling and segmentation algorithm on motions, and thus changes in the scene. Therefore, we actively manipulate objects. Before and after each manipulation a RGB-D sensor provides a point cloud of the environment. Due to the motion, the object causes changes in consecutive point clouds, which are used to track the object with the ICP algorithm. Furthermore, a three-dimensional object model can be built by combining the changes in the point clouds. Finally, the projection of the 3D model into the color image also allows a 2D segmentation.

3.1. Change Sets

Motion is a helpful tool for both segmentation and object modeling. In the context of point clouds that are produced by a statically mounted RGB-D sensor, the changes between two consecutive point clouds are influenced by motion. Under the assumption that only one rigid object moves between two consecutive point clouds, we consider the change to belong to this object. We use the set of changes for object modeling and also segmentation. In Chapter 4 we discuss, how an object must be moved to get a suitable change set between two point clouds. Figure 3.1 shows an example of such a change.

More formally, we have two point clouds $\mathcal{P}$ (old view) and $\mathcal{Q}$ (new view). A change set contains the changes between two point clouds, where a change is measured between two corresponding points in $\mathcal{P}$ and $\mathcal{Q}$. Since the RGB-D sensor is assumed to be static, corresponding points can easily be determined



(a) View before motion *(b)* View after motion

Figure 3.1.: The figure shows the change sets between two consecutive frames. In the old view the disappearing points are marked (blue), while new view contains the appearing points (red).

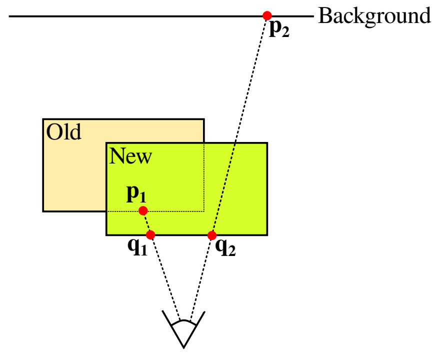
by their location in the depth image. In the following, two corresponding points are denoted as $\mathbf{p}$ and $\mathbf{q}$. We define the z-coordinate p_z of a point $\mathbf{p}$ as the depth of the point, i.e., its distance to the sensor center. Finally, we get two change sets. The first one, $\mathcal{C}_-$, contains the points that belong to the old view (*disappearing change set*), while the second one, $\mathcal{C}_+$, contains the points of the new view (*appearing change set*).

There are three possible options if we only regard the structure of the scene, i.e., its depth:

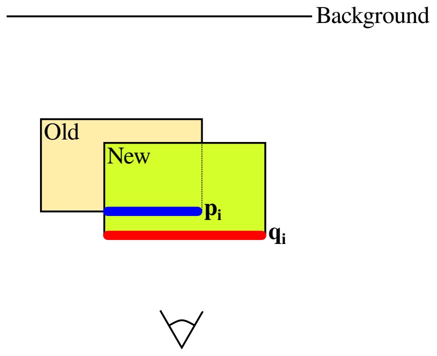
$p_z = q_z$**:** $\mathbf{p}$ and $\mathbf{q}$ are equal. Thus, such points do not contribute to a change set, since we assume that no sensible motion happened.

$p_z > q_z$**:** The depth of the new point is lower than the corresponding old depth. The membership of points in a change set is determined in the following way:

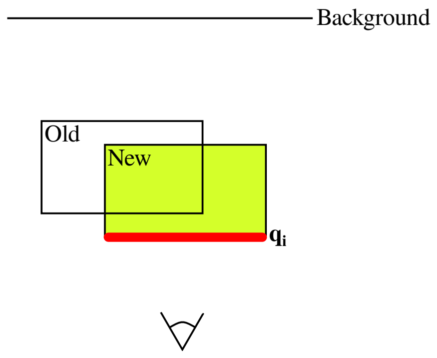
- *New point* $\mathbf{q}$: The new point is closer to the camera. In this case the new point can always be added to the change set of appearing points



(a) We have to take care of old points ($\mathbf{p}_1$, $\mathbf{p}_2$). They need not to be part of the moved object ($\mathbf{p}_1$) – they also can belong to the background ($\mathbf{p}_2$).

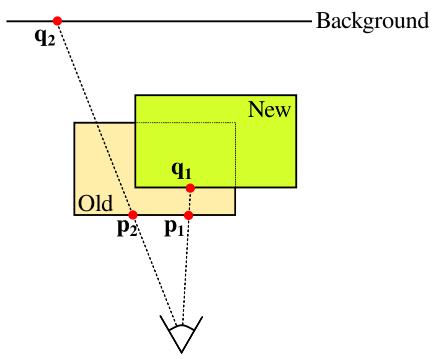


(b) An old point $\mathbf{p}_i$ can be added to the disappearing change set if it already belongs to the object in the old view. A new point $\mathbf{q}_i$ is always part of the appearing change set.

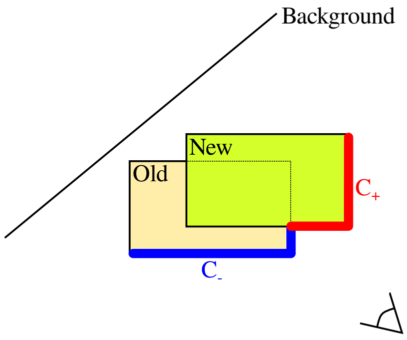


(c) If the old point $\mathbf{p}_i$ does not belong to a known object, we cannot be sure if the old point is background or not. Thus, it is not accounted for in the disappearing change set.

Figure 3.2.: The figure illustrates how we perform the change set computation between two point clouds if the new depth of a point is lower than the corresponding old one. All points with a new depth that is lower than the old one belong to the object, and thus they are added to the appearing change set. If we look at the corresponding old points we have to distinguish between points belonging to the background and points, which are part of the moved object. If the object never moved before, we cannot be sure whether the old point is part of the object or belongs to the background.

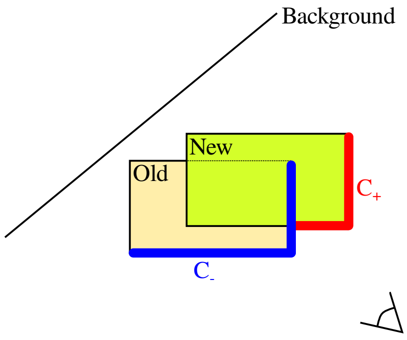


(a) We cannot distinguish if a point in the new view belongs to the background or to the object ($\mathbf{q}_1$ versus $\mathbf{q}_2$).

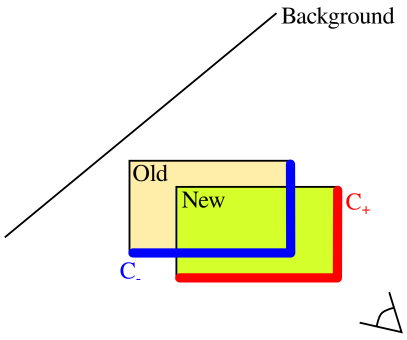


(b) Only the old points $\mathbf{p}_i$ (blue) are added to the disappearing change set.

Figure 3.3.: If the old depth of a point is lower than the corresponding new one, only the point of the old cloud can be added to the disappearing change set. No points are added to the appearing change set, since we cannot distinguish between background and object in the new view – it is still unknown. Whether a point belongs to the object will be determined later in the ICP step (see Section 3.2).



(a) In this example, all visible points of the old view belong to the disappearing change set (blue). From the appearing points (red) only those are part of the appearing change set, whose depth is lower than the old one.



(b) All points can be considered.

Figure 3.4.: Examples of change sets: Blue corresponds to disappearing points and red points appear in the new view.

$\mathcal{C}_+$. Figure 3.2 visualizes these points (marked with $\mathbf{q}_i$).

- *Old point* $\mathbf{p}$: The old point needs more differentiation (see Figure 3.2 (a) for an illustration). The old point can also be part of the background. Since we are interested in modeling a single object and remove the background from it, the old point belongs only to the change set of disappearing points if $\mathbf{p}$ was already a part of the object in the old view. Thus, if the old point belongs to the object we can be sure that it is not part of the background. All other points do not belong to $\mathcal{C}_-$ (see Figure 3.2 (c)).

$p_z < q_z$**:** The new point lies farther away from the sensor than the old point. The corresponding points are added to the change sets as follows (Figure Figure 3.3):

- *New point* $\mathbf{q}$: Whether a point in the new view belongs to the background or the object will be estimated later by ICP. Therefore, such points are ignored in the change set computation. They are not part of the appearing change set.

- *Old point* $\mathbf{p}$: The old point is closer to the camera. It can always be added to the change set of disappearing points $\mathcal{C}_-$. (see Figure 3.3 (b)).

If depth is equal, it is also possible to add visual changes (e.g., color). In doing so, the change set's size will increase. Thus, it is getting easier to match the change set against a point cloud with ICP (see Section 3.2). However, in practice, there are problems with light changes and shadows. We can avoid this problem by taking the depth-only change set and make sure to move the object far enough (see Chapter 4). Two examples for change sets are shown in Figure 3.4.

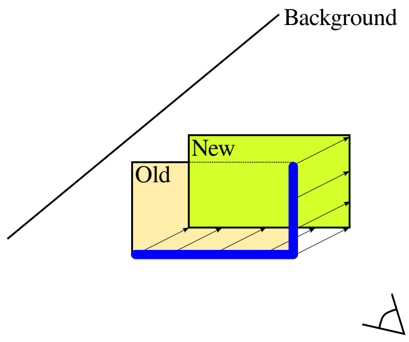


Figure 3.5.: The disappearing change set is matched against the new view. Based on this transformation we update the object model.

3.2. Object Tracking and Modeling

Tracking an object means to estimate its motion. There are various methods to track an object. It can be done with the ICP algorithm, which calculates the transformation between two point clouds. As mentioned, we move a single object in a scene. Between each motion a point cloud of the current scene is provided by a RGB-D sensor. Based on two point clouds we compute the change sets (see Section 3.1). These can be used to estimate the motion, i.e., the transformation of the object between two views. Figure 3.5 illustrates the transformation.

In our approach we use ICP to estimate the transformation of a change set to a point cloud. Let $\mathcal{P}$ and $\mathcal{Q}$ be two points clouds, where $\mathcal{P}$ is assumed to be the older one. Then, the change sets are given by $\mathcal{C}_+$ and $\mathcal{C}_-$. $\mathcal{C}_+$ contains a subset of points of $\mathcal{Q}$, whereas $\mathcal{C}_-$ is a subset of $\mathcal{P}$.

There are two possible ways to estimate the transformation. Either $\mathcal{C}_+$ is matched against the old view $\mathcal{P}$ or vice versa – matching $\mathcal{C}_-$ against $\mathcal{Q}$. In practice, we can compute both transformations to have a simple indicator to discard incorrect estimated transformations. They should be inverse to each other. However, it is advantageous to estimate the transformation from the disappearing change set $\mathcal{C}_-$ to the new view $\mathcal{Q}$. In general, compared to $\mathcal{C}_+$, $\mathcal{C}_-$ contains more points for matching (see Section 3.1).

Let $\mathbf{T} \in \mathbb{R}^{4\times 4}$ be a homogeneous transformation matrix that contains the estimated translation and rotation of the object. Since $\mathbf{T}$ is homogeneous, the points in the following equation are considered to be homogeneous, too, i.e., $\mathbf{p} = (x, y, z, 1)$. We compute the three-dimensional model $\mathcal{O}_n$ of the object by combining the change sets and the current model.

Therefore, let

$$\mathcal{P}^{\mathbf{T}} = \left\{\mathbf{T} \cdot \mathbf{p}_1, \mathbf{T} \cdot \mathbf{p}_2, ..., \mathbf{T} \cdot \mathbf{p}_{|\mathcal{P}|}\right\} \tag{3.1}$$

be a transformed point cloud. After n object movements we get the new object model by

$$\mathcal{O}_n = \mathcal{O}_{n-1}^{\mathbf{T}} \cup \mathcal{C}_{-}^{\mathbf{T}} \cup \mathcal{C}_{+}. \tag{3.2}$$

The more geometric features the object has, the better ICP will work. However, there is a problem whenever the change set contains too few points. Such a small change set can lead to a transformation estimation of poor quality. To avoid this, we can compute an enlarged change set by considering other views. For example, if the appearing change set $\mathcal{C}_{+}$ between the old view n and new view m is required, we can choose an arbitrary view as the old view (i.e., replacing view n by some other view) to compute $\mathcal{C}_{+}$. Hence, the change set size can be maximized to improve transformation estimation and the object model quality.

Furthermore, it is also possible to maximize all change set sizes at the end of the registration process to optimize the final result.

3.3. Preprocessing of Point Cloud Data

Before beginning to map objects in the scene we need to prepare the data provided by the RGB-D sensor. This section is subdivided into the following problems:

1. Filter the point cloud $\mathcal{P}$ to remove sensor noise.

2. Use RANSAC to detect the surface plane: As mentioned in Section 2.1

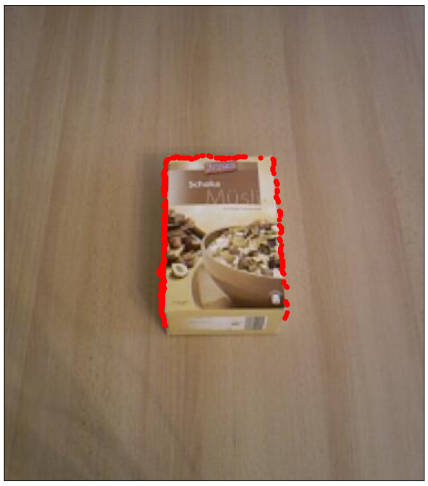


Figure 3.6.: Noisy points of a point cloud: The red dots correspond to points, which have an above-average variance in depth across several sensor readings. We ignore these points in further computation steps to avoid false positives in the change sets.

the objects are assumed to be placed on a planar surface. We use the RANSAC algorithm (see Section B.2) to detect this surface in the point cloud. As a result, we get a two-dimensional plane in the 3D space. The estimation is necessary, since the following planning and pushing problem is solved in 2D on this plane.

Noisy points in a point cloud can be a problem since they are wrongly added to the change sets. They occur on object edges, where sensor readings jump between different depth values. Figure 3.6 shows an example of such points. Pushing an object results in the expected change in the point cloud plus some noise at edges of other objects. If we do not account for these points the assumption that only one object was moved simultaneously cannot be fulfilled anymore – we may have changes at every object.

Assuming a statically mounted RGB-D sensor, corresponding points in different point clouds can be determined by their location in the depth image. To address the sensor noise we measure the variance in depth at each point over a number of frames. The variance in depth is an adequate measure to detect the jumping sensor readings. Theoretically, the depth of each point should not change if the scene is static and the RGB-D sensor does not move. Hence, the depth variance is under a threshold ϵ if the point does not change.

We are interested in computing the mean depth and the variance of all depth values of n collected point clouds. They can be determined using the following estimator [50] of the expected value:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} z_i, \tag{3.3}$$

where $z_i \in \mathbb{R}$ is the depth of a point in the i-th view. The estimator for the corresponding variance is then given by

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n} \left(z_i - \mu\right)^2 . \tag{3.4}$$

All points, which have an estimated variance $\sigma^2 > \epsilon$ are rejected from further consideration.

3.4. Image Segmentation

Segmenting an object from the background is important for generating training data for, e.g., an object detector. The 3D object model, which is generated from each view can be regarded as a three-dimensional segmentation into object and background (every point of the scene, which does not belong to the object is background). Just as segmenting in the three-dimensional space, we can segment the color or depth image that is also provided from the RGB-D sensor. Applying a projection of the current 3D object model into the images yields the two-dimensional segmentation. Figure 3.7 shows an example of the resulting segmentation of the color image.

Using the simplest camera model, namely the pinhole camera model [17], the projection of a point $\mathbf{X} = (X, Y, Z)$ to a point $\mathbf{x} = (x, y)$ on the image

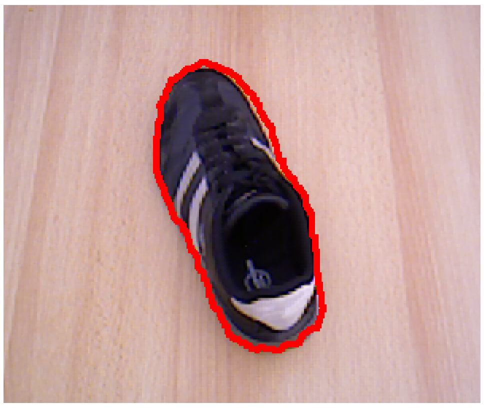
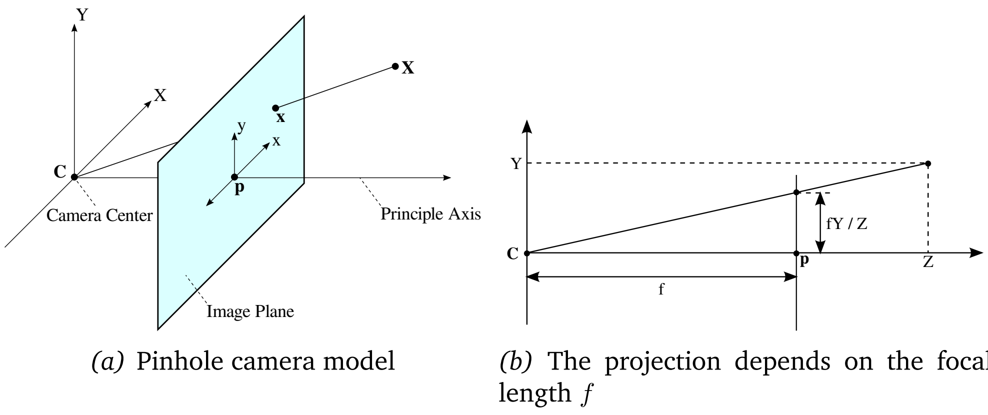


Figure 3.7.: The projection of the current object model into the color image yields the segmentation of the image.



(a) Pinhole camera model *(b)* The projection depends on the focal length f

Figure 3.8.: The pinhole camera model describes the projection of a 3D point to a plane.

plane is done by

$$x = f \cdot \frac{X}{Z} \quad \text{and} \quad y = f \cdot \frac{Y}{Z}. \tag{3.5}$$

Figure 3.8 (a) and (b) illustrate how the projection works.

To allow a better approximation of reality additional parameters are considered for the projection. They are named *intrinsic* camera parameters: the location of the principle point $\mathbf{p}$ (location of the center in the image plane), the size ratio of a pixel denoted by m_x and m_y and the skew parameter s, which is zero for most cameras.

Based on the above camera model a more general projection matrix $\mathbf{P}$ can be defined by

$$\mathbf{P} = \mathbf{K}\mathbf{M}, \tag{3.6}$$

where

$$\mathbf{K} = \begin{pmatrix} m_x \cdot f & s & p_x \\ 0 & m_y \cdot f & p_y \\ 0 & 0 & 1 \end{pmatrix} \tag{3.7}$$

is the intrinsic camera matrix, which contains the internal camera parameters. Furthermore, the matrix

$$\mathbf{M} = (\mathbf{R} \,|\, \mathbf{t}) \tag{3.8}$$

describes the pose of the sensor in relation to a global coordinate system, where $\mathbf{R} \in \mathbb{R}^{3\times3}$ and $\mathbf{t} \in \mathbb{R}^3$ are the rotation and translation, respectively. Based on Equation (3.6) the object model $\mathcal{O}$ can be projected into the color or depth image, which yields the 2D segmentation:

$$\mathbf{x} = \mathbf{P} \cdot \mathbf{p}. \tag{3.9}$$

Summary

We use motion to segment objects from the background in a point cloud and color image provided by a RGB-D sensor. When moved, the object causes a change in the point cloud. We denote the appearing and disappearing points as *appearing change set* and *disappearing change set*, respectively. They are computed using the presented rules. We build a three-dimensional object model by combining several change sets. The object's transformation between two views is estimated by ICP. Due to sensor noise it is important to filter out points, which have a high variance in depth. These points mostly occur on edges, where points jump between different depth levels. Furthermore, the planar working surface is estimated by RANSAC, which is necessary for the following planning of manipulation actions. Finally, the 3D object model can be also used to do a 2D segmentation of the color and depth image using a projection.

4 Pushing Objects

Autonomous object modeling and segmentation by scene manipulation needs a manipulator that is able to push or grasp items. We propose manipulation by pushing since accurate grasping requires in-depth knowledge about the object, e.g., a spatial object model. Based on the idea to start exploration without any prior knowledge of the scene it is therefore easier to use pushing than grasping. Pushing also allows removing the manipulator from the scene before acquiring sensor data. Hence, we do not have to filter out the manipulator from the sensor data.

There are different ways and purposes for manipulating objects. We distinguish between an initial manipulation and an informed object interaction. The initial object manipulations are used to determine movable object candidates using a heuristic. Then, the robot tries to manipulate the object candidate. If no moving object is detected, we assume that it belongs to the environment, is too heavy or is blocked by another object or by the environment. On the other side, if motion takes place, we can regard the object as movable, i.e., the object candidate will be further explored. Based on the manipulations, we build a 3D object model using the algorithm presented in Chapter 3. Therefore, the robot manipulates the object in such a way that complete object modeling is possible. The planning and pushing algorithms use the continuously updated object model to improve the quality of the manipulations. Finally, we show how to execute an intended pushing manipulation with a robotic gripper. We use a two-finger-gripper to allow stable manipulations.

4.1. Using an Object Model for Manipulation

In an unknown scene, we only have partial information – there is only what we can see in the camera image. We also do not know if some part of the scene is an object or background. The parts of the scene that are occluded cannot be used to decide which operation to do next. By regarding all views collected so far, we can build a 3D representation of an object (see Chapter 3) using the object's motion. Each object manipulation improves the model by insertions of new views. We will first consider the case where a partial object model is already given, i.e., we know where the object is. Afterwards, in Section 4.2, we discuss, how object candidates can be found in the scene.

The object model is used to plan the manipulations. The planning step computes the next preferred pose of the object based on the object model generated so far. We plan a collision free path from the current pose to the goal pose to avoid collisions with other objects or the scene.

We set up a configuration space, which contains discretized object configurations. A* searches a path from the current object pose to the goal pose in the configuration space of the object.

4.1.1. Object Configuration Space

In the described setup we push objects on a planar surface. This implies that the necessary transformation is a three-dimensional one. A configuration

$$\mathbf{c} = (x, y, \theta) \tag{4.1}$$

is therefore given by its location x, y and the orientation θ.

Let the configuration space of an object be given by

$$\mathcal{C} \in \mathbb{R}^2 \times [0, 2\pi), \tag{4.2}$$

where each configuration $\mathbf{c}$ corresponds to a potential value. The intuition behind the potentials is that they describe the quality of an object configuration. Thus, configurations with a high quality are the preferred ones in

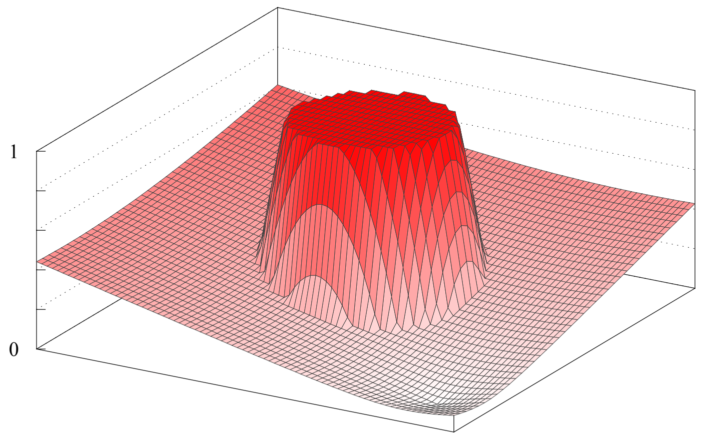


Figure 4.1.: Example for a combined, attractive and repulsive 2D potential field. It was clamped to a maximum value for illustration.

the path planning algorithm. We plan a path from the current pose to the goal pose using the potentials. The planning algorithm uses the potentials as a cost function. The configuration with the lowest potential corresponds to the current goal configuration. Finally, the manipulator executes the computed path. A formal definition of potential fields can be found in Appendix B.3. Figure 4.1 shows an example of a potential field.

The potential function $U(\mathbf{c})$ assigns a potential value to a configuration $\mathbf{c}$. It is computed by summation over the repulsive and attractive potential. The repulsive potentials correspond to configurations, which are more likely to be blocked (e.g., a configuration that is out of reach of the manipulator gets a high potential). On the other hand, attractive potentials correspond to possible goal configurations. We will now show how the potential field $U(\mathbf{c})$ is defined in our setup.

Repulsive Potential: Working Area of the Robot and Objects A robot normally has a constrained working area. In general, each configuration of the regarded object that lies outside of this working area (on the planar surface) has to be neglected as a potential goal configuration. We set the potential of those configurations to a high constant value which results in a repulsive potential.

Additionally, all objects in the working area of a robot must be accounted for. These are all objects, which are visible in the current point cloud. For the potential we assume that all points, which lie above the working surface belong to an object. But we also have the information about known objects, which the robot already has moved and partially modeled. The known object models may contain more information about the occluded areas. Thus, they are completely considered as a repulsive potential.

Consequently, we define the potentials of inadmissible configurations by a high value d_{max}. All other configurations get a potential depending on the inverse distance to the closest repulsive potential:

$$U_{-}(\mathbf{c}) = \begin{cases} d_{\text{max}}, & \mathbf{c} \text{ not possible} \\ d_{\text{max}} - d, & \text{else} \end{cases}, \tag{4.3}$$

where d is the Euclidean distance between $\mathbf{c}$ and the closest configuration that is not admissible. The effect of $U_{-}(\mathbf{c})$ is that inadmissible configurations get a high potential and will not be used in the planning step. On the other hand, admissible configurations get a lower potential and can be used for path planning. The larger the distance to an inadmissible configuration is, the lower and better the potential will be. Hence, the planning algorithm prefers configurations that correspond to locations at the center of the working area of the manipulator. It also avoids configurations that are in conflict with other objects.

Attractive Potential: Favored Translation and Rotation For the goal configurations we design an attractive potential. Hence, the configurations that correspond to the preferred distance and angle get an attractive potential. How far an object should be pushed depends on its size. A large object has to be pushed further than a small one to get a suitable change in the point cloud. If the change is too small, we cannot use ICP to match the change set against the point cloud of the current view, which means that the estimated transformation can be of poor quality (see Chapter 3). Thus, we need to determine the size of an object to find an appropriate distance for pushing.

We define the size as the object extents in the directions of the first two principle components (see Figure 4.2 (a) for an illustration). Let these axes be given by the direction vectors $\vec{\mathbf{o}}_1 \in \mathbb{R}^2$ and $\vec{\mathbf{o}}_2 \in \mathbb{R}^2$. The origin of the axes is the geometric center $\mathbf{m} \in \mathbb{R}^2$ (see Section 2.2). Based on the two axes, we can compute the length of the object in the corresponding directions. Projecting all object points onto both axes gives the size estimation. The projection is given by

$$\mathcal{P}_1 = \left\{ \frac{(\mathbf{p}_i - \mathbf{m}) \cdot \vec{\mathbf{o}}_1}{\vec{\mathbf{o}}_1 \cdot \vec{\mathbf{o}}_1} \cdot \vec{\mathbf{o}}_1 \,\middle|\, 0 \leq i \leq n \right\} \quad \text{and} \tag{4.4}$$

$$\mathcal{P}_2 = \left\{ \frac{(\mathbf{p}_i - \mathbf{m}) \cdot \vec{\mathbf{o}}_2}{\vec{\mathbf{o}}_2 \cdot \vec{\mathbf{o}}_2} \cdot \vec{\mathbf{o}}_2 \,\middle|\, 0 \leq i \leq n \right\}. \tag{4.5}$$

The size in each direction is then the maximum distance between two points on the axes:

$$l_1 = \max_{\mathbf{p}_{i,j} \in \mathcal{P}_1} \left\{ \|\mathbf{p}_i - \mathbf{p}_j\|_2 \right\} \quad \text{and} \tag{4.6}$$

$$l_2 = \max_{\mathbf{p}_{i,j} \in \mathcal{P}_2} \left\{ \|\mathbf{p}_i - \mathbf{p}_j\|_2 \right\}. \tag{4.7}$$

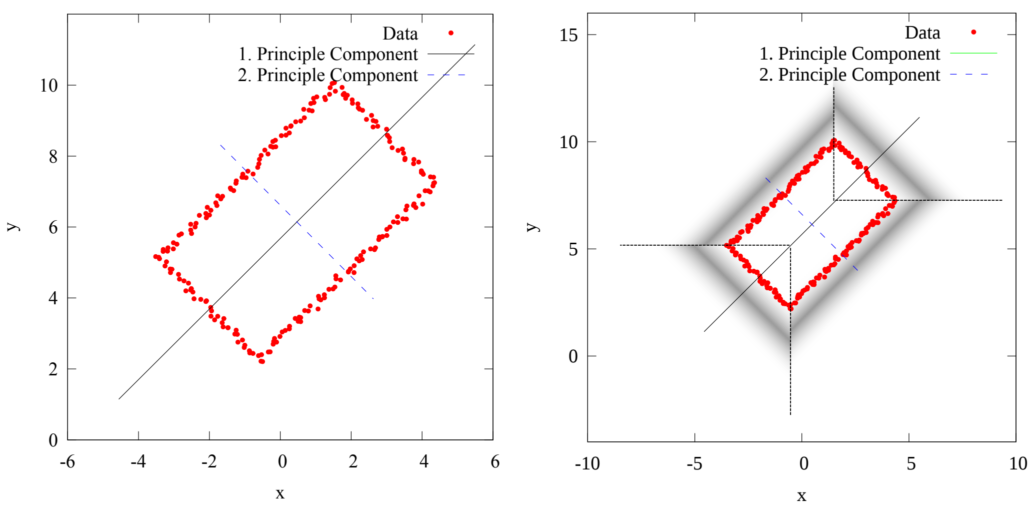
Depending on the angle between push direction and the axes we can choose

$$d_1^* = \xi_1 \cdot l_1 \quad \text{or} \quad d_2^* = \xi_1 \cdot l_2 \tag{4.8}$$

as the push distance. ξ_1 and ξ_2 are weighting parameters. Setting ξ_1 to 0.5 would mean pushing 50% of the object extent in the direction of the first axis. To allow more flexibility in planning, d_i^* is the mean of a normal distribution with a user defined standard deviation of σ:

$$U_+^{\text{dist}}(\mathbf{c}) = -\frac{1}{\sigma\sqrt{2\pi}} \exp\left\{ -\frac{1}{2} \cdot \left(\frac{d((x,y), o_i) - d_i^*}{\sigma} \right)^2 \right\}. \tag{4.9}$$

$d((x,y), o_i)$ is the orthogonal distance between the configuration position and the corresponding axis o_i. Figure 4.2 (b) illustrates how the potential is constructed around the object using Equation (4.9).



(a) We apply PCA to a point set to estimate the extent of the corresponding object

(b) Attractive potential field of pushing distance: dark gray corresponds to the lowest potential, i.e., preferred goal positions

Figure 4.2.: The two principle components of an object are used to estimate its extent. Based on the extent we compute the attractive potential field of the preferred pushing distance. A larger change set improves the transformation quality of ICP.

To map all sides of the object the robot also needs to rotate it since the camera is assumed to be static. We decided to rotate the object by a user defined value θ^*. θ^* is the mean of a normal distribution with a standard deviation of σ, too:

$$U_+^{\text{rot}}(\mathbf{c}) = -\frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{1}{2} \cdot \left(\frac{\theta - \theta^*}{\sigma}\right)^2\right\}. \qquad (4.10)$$

$\theta = 0$ means no rotation of the object.

Summing up all potentials yields the final potential for a configuration, which is used for path planning.

4.1.2. Planning a Collision Free Path

In the context of small object movements path planning plays a minor role, since most of the push operations can be performed directly without any intermediate step. Nonetheless, in general, a complex movement with intermediate steps is possible. To avoid a local minimum as a goal configuration we do not use gradient descent based planning (see Appendix B.3 for details). Instead we use A* for planning (Appendix B.4).

First, we need to choose a goal configuration from $\mathcal{C}$. Attractive potentials are low and thus appropriate goal configurations. It makes sense to use the configuration with the minimum potential value as the goal configuration.

The distance function between two configurations is given by

$$d(\mathbf{c}_1, \mathbf{c}_2) = \left|\left| \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} - \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \right|\right|_1 \cdot U(\mathbf{c}_2) \tag{4.11}$$

Our heuristic

$$h(\mathbf{c}) = ||\mathbf{c} - \mathbf{c}_{\text{goal}}||_1 \cdot U(\mathbf{c}_{\text{goal}}) \tag{4.12}$$

is admissible, since in a grid the Manhattan distance is admissible and $U(\mathbf{c}_{\text{goal}})$ is the minimum of all potential values.

4.2. Exploration of Unknown Objects

So far, we only considered the case where an object model is already given for manipulation planning. Unknown objects require an initial manipulation to get a first object model, which is then used for further processing (see Section 4.1).

Whenever the robot needs to explore unknown parts of the environment it carries out initial push operations. As there is no in-depth information about the objects in the scene, we use the point cloud data to find possible object candidates and locations where the manipulator can push them. To avoid that the robot slips off the object at edges resulting in difficult to predict object movements, it should push objects at the center points of surfaces.

The normals of the point cloud are used to find potential pushing locations in the scene. We use a region growing algorithm to detect appropriate surface patches where the robot can push. As a result we get regions with homogeneous normals. The region growing algorithm works as follows:

1. Compute the normals $\vec{\mathbf{n}}_i$ of the current point cloud $\mathcal{P} = \{\mathbf{p}_1, ..., \mathbf{p}_n\}$.

2. Choose an arbitrary point $\mathbf{p}_i$ and its normal $\vec{\mathbf{n}}_i$ as a start point.

3. Find the set of neighbors $\mathcal{P}_i$ and corresponding normals $\mathcal{N}_i$ of $\mathbf{p}_i$.

4. To be more robust against outliers we compute the mean of the neighbor normals:

$$\vec{\mu} = \frac{1}{|\mathcal{N}_i|} \cdot \sum_{\vec{\mathbf{n}} \in \mathcal{N}_i} \vec{\mathbf{n}}. \tag{4.13}$$

5. Recursively, find neighbors and compute in each case the angle between $\vec{\mu}$ and the normals in the neighborhood. The point belongs to the region, seeded by $\mathbf{p}_i$, if the angle between $\vec{\mu}$ and a neighbor normal $\vec{\mathbf{n}}_p$ is lower than a threshold ϵ:

$$\arccos\left(\frac{\vec{\mu} \cdot \vec{\mathbf{n}}_q}{\|\vec{\mu}\|_2 \cdot \|\vec{\mathbf{n}}_q\|_2}\right) \leq \epsilon. \tag{4.14}$$

   The recursion stops if no more neighbors can fulfill Equation (4.14).

6. Repeat the steps 2) to 5) until all points were processed.

To find a suitable surface where the robot can push we choose the largest region (number of points belonging to it). However, we must account for the normal direction. Regions with normals that are orthogonal to the working surface normal are more suitable to push than normals pointing upwards. Thus, we choose the largest region but neglecting regions with normals parallel to the working surface normal. The center of the selected region is then used for pushing. Since there is no in-depth information of the object the direction is set in such a way that the robot pushes the object in the inverse normal direction. Both location and direction are used to execute a linear push operation as explained in the next section.

4.3. Object Manipulation

Finally, we present how the objects can be moved if the goal location has been decided. We assume that the robot's gripper consists of two fingers. In the planning step they are regarded as points. Rotational and translational movements are considered separately to allow more accurate movements. The planning problem is considered to be a two-dimensional one. Both translational and rotational manipulation requires the projection of the 3D object model to the working surface. Afterwards, the concave hull of the projected points is computed. This can be done using α*-shapes* (see Appendix B.6 for details). Let the concave hull be given by an ordered set of line segments

$$\mathcal{H} = \left\{ \left(\mathbf{s}_1, \mathbf{e}_1\right), ..., \left(\mathbf{s}_n, \mathbf{e}_n\right) \right\}, \tag{4.15}$$

where $\mathbf{s}_i \in \mathbb{R}^2$ is the start point and $\mathbf{e}_i \in \mathbb{R}^2$ the end point of the corresponding line segment.

4.3.1. Physical Assumptions

It is not possible to compute the motion of an object without knowledge of the support friction $S(\mathbf{x})$. The support friction is the location-dependent friction between object and working surface. Both translational and rotational movement are strongly dependent on it because the center of rotation will differ if the support friction changes. In real applications it is difficult to estimate $S(\mathbf{x})$.

For a known support friction the center of rotation can be computed [35]. It is also possible to learn how an object will move given a set of push actions as proposed by Omrcen et al. [34]. However, their learning approach needs approximately 50 push actions for convergence. We approximate the center of rotation by the geometric center. In doing so, the rotational push actions will always cause a translational movement. However, in our setup it is adequate to use this approximation, since we need to map the whole object. In the experiments it will be shown, that the executed rotational push action approximates the planned rotation quite well.

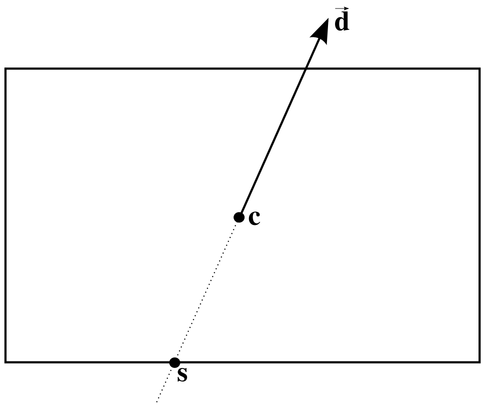


Figure 4.3.: The robot places the gripper at the push point $\mathbf{s}$ and pushes the object in the direction $\vec{\mathbf{d}}$. $\mathbf{c}$ is the geometric object center. The rectangle illustrates the object.

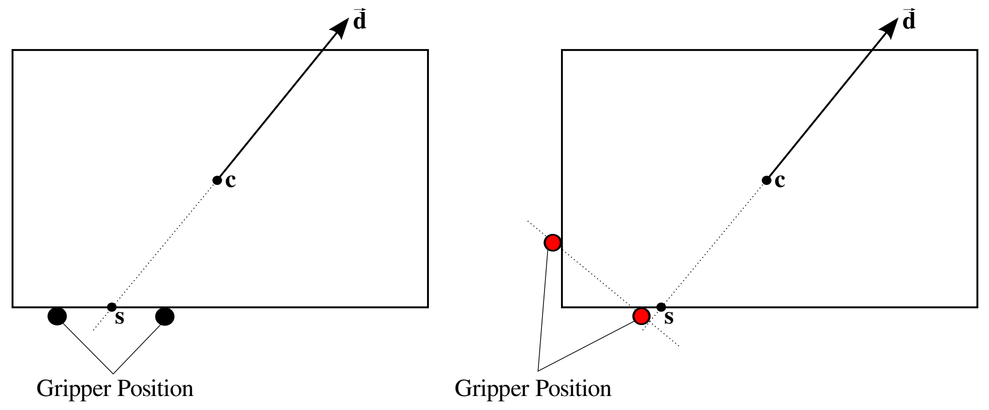
Furthermore, all pushing forces lie in the horizontal plane and the gravity acts in vertical direction. Both the manipulator and object move in the horizontal plane. The friction is uniformly distributed over the support plane between object and working surface. In addition, pushing motions are executed slowly enough such that inertial forces can be neglected. This is called *quasi-static* assumption, i.e., pushing forces are balanced by the frictional forces in the support plane.

4.3.2. Translational Manipulation

The push point of a translational push action is set in such a way that the force is acting into the direction of the center of rotation. As denoted earlier, we approximate the center of rotation by the geometric center. The push direction and distance are determined in the planning step that is explained in Section 4.1 and 4.2.

Let the push direction be given as $\vec{\mathbf{d}} \in \mathbb{R}^2$. We have to find the intersection of the object border with the line that is defined by the geometric center $\mathbf{c} \in \mathbb{R}^2$ and the opposite push direction to get a stable push action. Figure 4.3 illustrates the computation of how the push point $\mathbf{s} \in \mathbb{R}^2$ is computed.

Based on this point the object can be pushed (see Figure 4.4 (a)). Such



(a) Non-stable manipulation *(b)* Stable pushing manipulation

Figure 4.4.: To allow a more stable motion of the object we propose to move the gripper to a position where the gripper fingers are placed orthogonal to the pushing direction $\vec{\mathbf{d}}$. $\mathbf{c}$ is the geometric object center.

push actions can cause a drift between gripper and object – the object may move in the wrong direction. We propose to place the gripper in such a way that it is orthogonal to the pushing direction. Therefore, it is necessary to move the push point on the edge of the object to find an adequate position (see Figure 4.4 (b)).

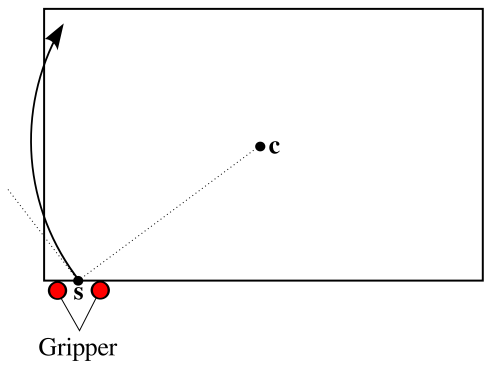
The full algorithm consists of the following steps:

1. Project all object points onto the working surface given by RANSAC.

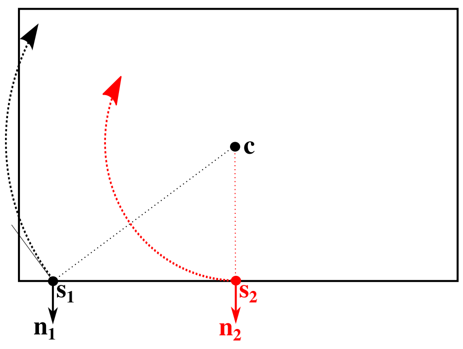
2. Compute the concave hull $\mathcal{H}$, i.e., the shape of the object (α-shapes).

3. Find the intersection of the opposite pushing direction (a line segment that starts at the geometric center and has the opposite pushing direction) with the shape. Since there can be multiple points in concave objects that has an intersection with this line, we choose the one, which has the largest distance to the geometric center. This is reasonable because the other intersection points may lie at difficult to reach positions. This intersection is the pushing point.

4. The pushing point can be moved to find a better position, which leads



(a) Rotational pushing



(b) Where should the rotational pushing start? s_1 is the favored pushing point.

Figure 4.5.: The rotational pushing is done by moving the gripper on a on an arc around the geometric center. The pushing starts at a point where the normal and pushing direction point in preferably opposite directions to avoid drifting between object and gripper.

to a more stable movement. Therefore, the point is moved on the border until a position is found that allows the gripper to be placed orthogonally to the pushing direction. Also, both fingers have to be in contact with the object at this location. The maximum distance for moving the pushing point is bounded by a threshold. Otherwise, it is not moved.

5. For execution we move the gripper to the pushing point and then linearly in the pushing direction until the required distance is reached.

4.3.3. Rotational Manipulation

Rotating an object is important for object modeling. It is done by placing the gripper at a location on the edge of the object. Then, the gripper is moved on an arch around the geometric center (Figure 4.5 (a)). To reduce drifting between object and gripper, the pushing direction and object normal should point in opposite directions (Figure 4.5 (b)). In this example, s_1 is preferred over s_2. In contrast to s_1, s_2 is a bad pushing point since pushing on an arc around the geometric center is difficult at this location.

The algorithm for rotational pushing consists of the following steps:

1. Project all object points onto the working surface given by RANSAC.
2. Compute the concave hull $\mathcal{H}$, i.e., the shape of the object (α-shapes).
3. Based on the ordered set of the line segments of the concave hull we compute the shape normals $\vec{\mathbf{n}}_i \in \mathbb{R}^2$.
4. Choose the point with the largest angle between the normal and push direction. The push direction is assumed to be perpendicular to $(\mathbf{c} - \mathbf{s})$ at the beginning. To avoid that the gripper slips off the object at corners a minimum distance to the end of the corresponding line segment in $\mathcal{H}$ is enforced.
5. Finally, the execution of the rotation is done by moving the gripper on a circular path around the geometric center of the object that starts at $\mathbf{s}$.

Summary

We use a robotic manipulator for manipulating objects on the working surface. Therefore, the object modeling algorithm, which we describe in Chapter 3 computes a 3D model of the object using its motion. Based on the projection of this model to the working surface we construct a configuration space that contains possible configurations of the object. Potentials are assigned to each configuration that correspond to the quality of a configuration. There are two types of potentials: repulsive and attractive potentials. The repulsive potentials correspond to configurations that are not suitable (e.g., out of reach of the manipulator or colliding with other objects). On the other hand, attractive potentials describe preferred goal configurations. The final potential is the sum over both potential types. Then, the A* algorithm is used to find a path in the configuration space from the current pose to the pose with the lowest potential resulting in rotational and translational push actions. At the beginning, no object model is available. Thus, we do an initial manipulation. Region growing on the normals gives a set of regions with uniform normals. They are used to find an initial pushing point. Finally, we divide the manipulation in a translational and rotational pushing action resulting in robust pushing.

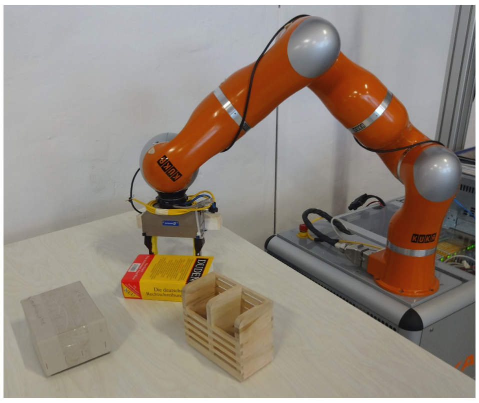
5 Evaluation

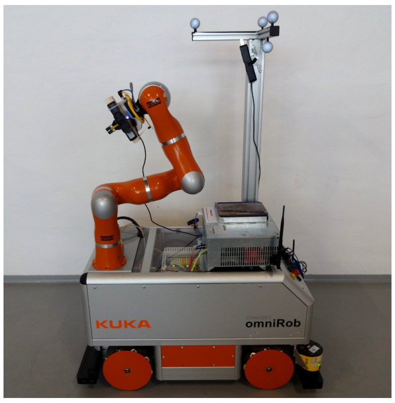
First, we describe the experimental setup. We shortly present the hardware, i.e., the manipulator, the RGB-D sensor and the robotic platform. Additionally, the implementation is presented. Afterwards, we evaluate and characterize the approach in a number of experiments.

5.1. Experimental Setup

We use the robotic platform *omniRob* from KUKA with a mounted manipulator, namely the *Light Weight Robot* (LWR). For stable push operations, a *Schunk WSG* gripper is mounted as an end-effector. For acquiring point clouds and color images we use the ASUS *XtionPRO LIVE* RGB-D sensor. We implemented our software using the *Robot Operating System* (ROS) and the *Point Cloud Library* (PCL).

We placed the objects on a table in front of the robot. The robot uses its manipulator to move the objects. Furthermore, we mounted the RGB-D sensor on the gripper. The sensors location remained static when taking new point clouds, i.e., after each manipulation the gripper returned to its sensor location. The manipulator is accurate enough that the assumption of a static placed sensor holds. Figure 5.1 (a) shows the setup. We use the ARToolkit in some experiments to detect marker poses [2].


(a) Experimental setup



(b) omniRob from KUKA

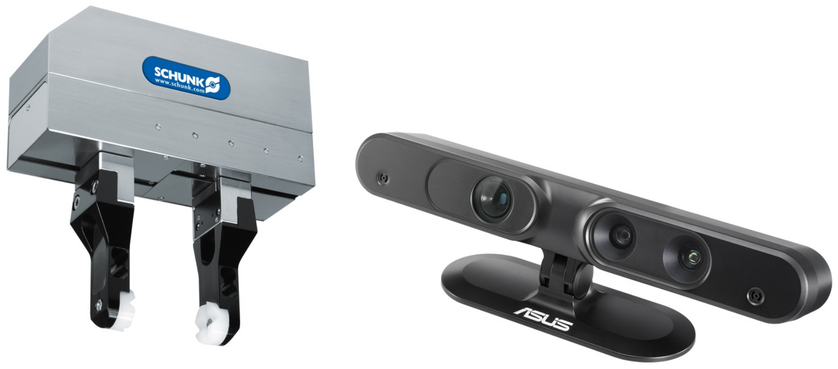
Figure 5.1.: We use an omniRob from KUKA with a mounted *Light Weight Robot*. The objects are placed on a table in front of the robot.

5.1.1. KUKA omniRob with Light Weight Robot

To perform the experiments we used a *omniRob* research robot from *KUKA Laboratories GmbH*, Augsburg (Germany). Figure 5.1 (b) shows an image of the robot. The platform itself has dimensions of 1.15 m $\times$ 0.86 m and a total weight of approximately 250 kg. It is equipped with two laser range finders. Four independently powered *omniMove* wheels allow an omni-directional movement. We use the platform only for controlling the manipulator – it was not moved for the experiments. In our experiments the mounted *Light Weight Robot* (LWR) is used for pushing the objects. It has seven degrees of freedom. The manipulator is capable of measuring the torques in each joint and therefore the external forces acting on the end-effector. A tool, for example a gripper or a hand can additionally be mounted to the LWR.

5.1.2. Schunk WSG Gripper

A gripper was mounted as an end-effector of the manipulator for the push operations. We used a *WSG 50* gripper from *SCHUNK GmbH & Co. KG*, Germany (see Figure 5.2 (a)) [43]. It has two fingers with integrated force sensors. The force sensors allow a sensitive grasping of objects. A grasping force of 80 N allows to elevate objects with a maximum weight of 0.8 kg.



(a) Schunk WSG 50 [43]   *(b) ASUS XtionPRO LIVE* [22]

Figure 5.2.: The gripper allows stable pushing of objects. The RGB-D sensor from ASUS provides point clouds, color and depth images.

The maximum distance between both fingers is 105 mm. We only use the two fingers of the gripper to allow a stable push action. The force sensors were not used in our experiments.

5.1.3. ASUS XtionPRO LIVE

We use an *ASUS XtionPRO LIVE* RGB-D sensor to acquire point clouds and color images [21, 22]. Figure 5.2 (b) shows the RGB-D sensor, which is used for the experiments. An advantage of this sensor is its small size. Furthermore, no external power supply is needed. The operating range lies between 0.8 m and 3.5 m, while the angles of aperture are 58°, 45° and 70° (horizontal, vertical, diagonal, respectively). Beside color and depth images the camera can record audio signals via a stereo microphone.

5.1.4. Software

The software was implemented in C++ on the basis of the *Robot Operating System* (ROS) [38]. ROS is an open source library, which includes many useful tools for implementing software in the robotics context. Amongst others, it handles the communication between different ROS nodes (i.e., executables running based on ROS). We use the *Point Cloud Library* (PCL) [41] for

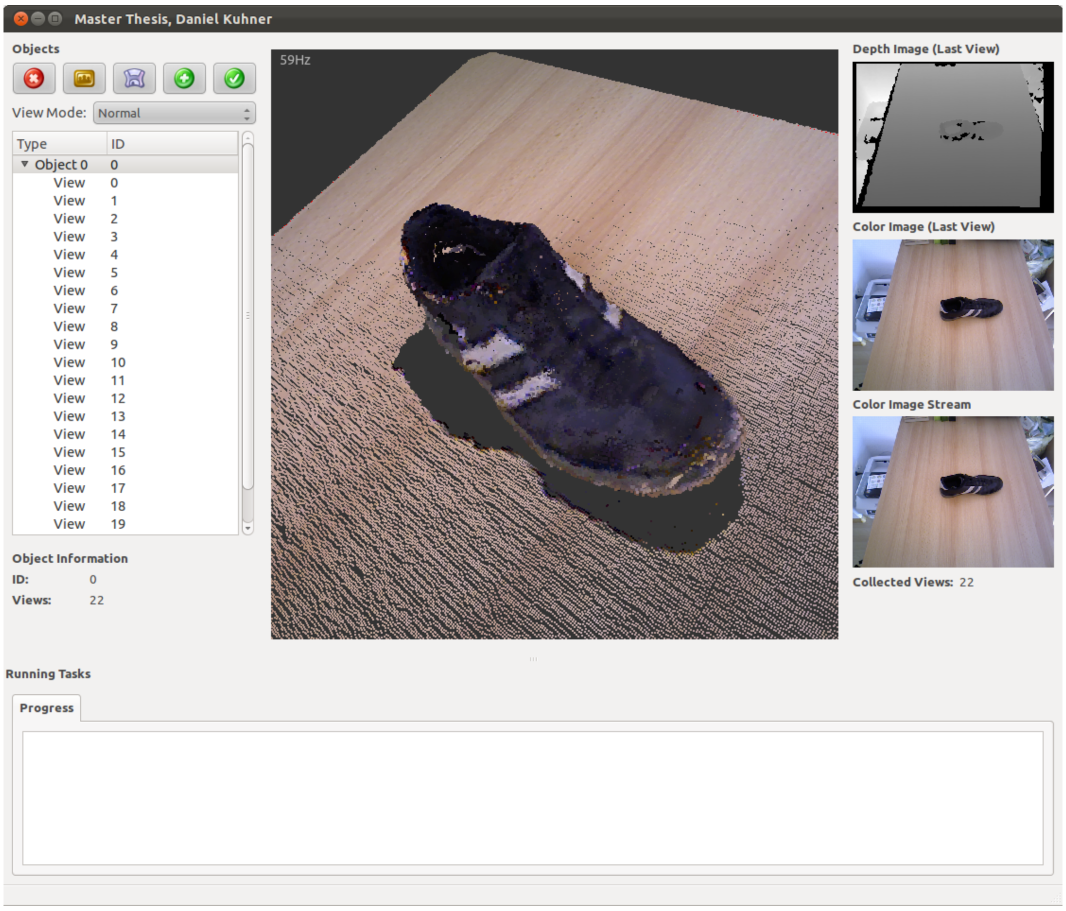
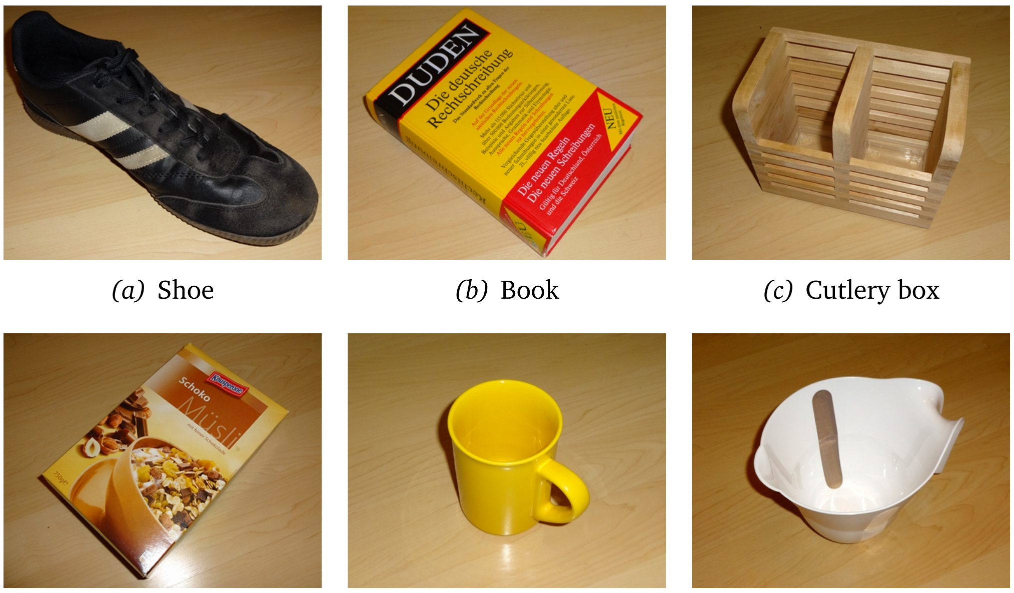


Figure 5.3.: User interface of our implementation

processing the point clouds generated by RGB-D sensor data. Additionally, the *RoboticsAPI* from KUKA is used for controlling the manipulator. Finally, we developed a graphical user interface that contains a 3D-viewer for visualization of the environment and the objects (see Figure 5.3).

5.2. Experiments

We evaluate the different aspects of our approach by a series of experiments. First, the quality of the object models is examined visually. Beside the visual results, we determine the accuracy of our model-based segmentation in a quantitative way. Afterwards, we present the results on the quality of



(a) Shoe *(b)* Book *(c)* Cutlery box

(d) Cereal box *(e)* Cup *(f)* Beaker

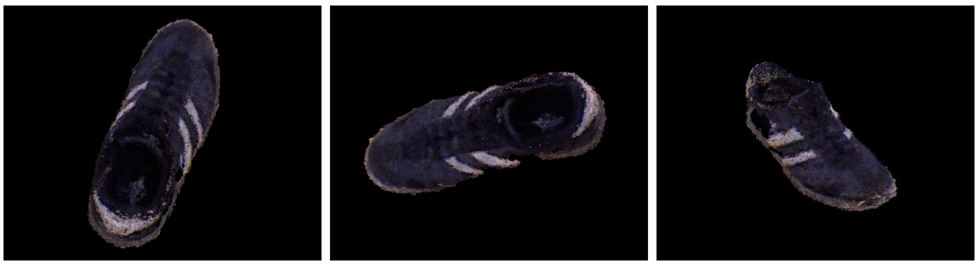
Figure 5.4.: Objects used in the experiments

our push manipulations. We also show, how the number of objects on the working surface influences the table estimation. Finally, the last experiment motivates and analyzes the filtering of jumping points in consecutive, static point clouds. Figure 5.4 shows the objects that were used in the experiments.

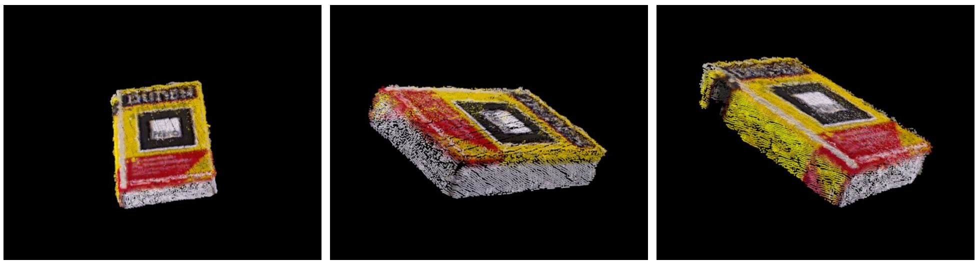
5.2.1. Quality of the Object Model

The presented approach is designed to generate training data for, e.g., object grasping by manipulating the objects in the scene. Figure 5.5 shows three views of resulting 3D object models. Objects with enough geometric features are modeled quite accurately.

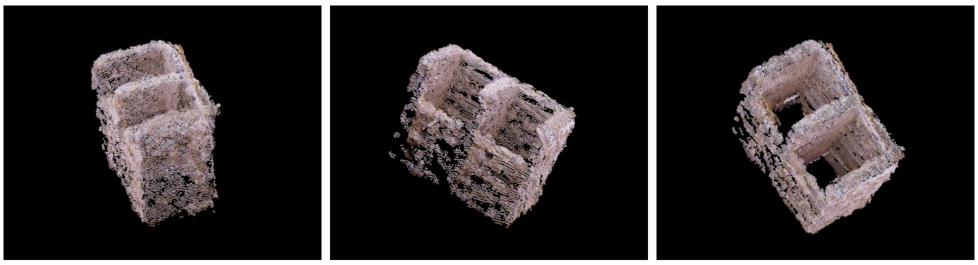
For predominantly rotationally symmetric objects, like the cup or the beaker shown in Figure 5.6, ICP does not match the handle in the correct way. The point cloud of these objects does not contain the complete surface. For instance, the point clouds of a cup acquired diagonally from above do not contain points from the left and right side of the cup. Thus, ICP will match a new view with respect to the missing side points and neglect the points of



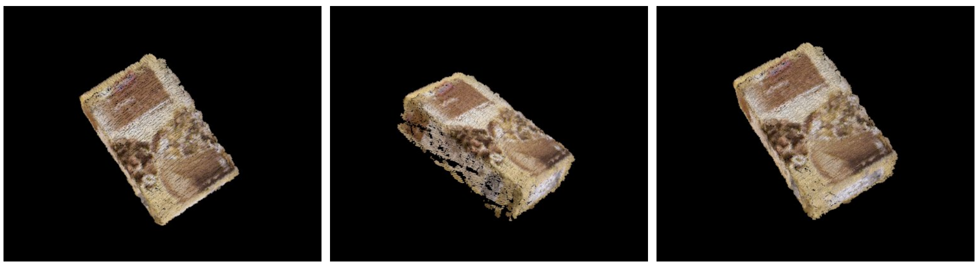
(a) Shoe



(b) Book



(c) Cutlery box



(d) Cereal box

Figure 5.5.: Some examples of the resulting point clouds that represent the 3D object models generated by our approach.

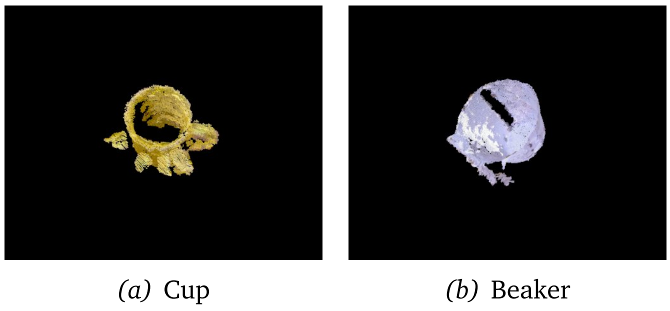


Figure 5.6.: ICP cannot match the single change sets of rotationally symmetric objects. In these two examples the handle was wrongly matched.

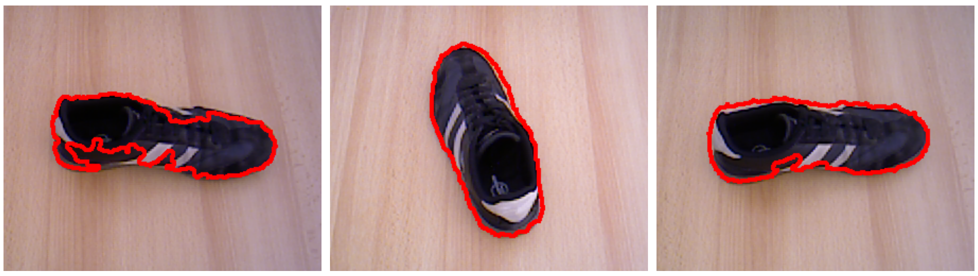
the handle, because of the comparatively low number of points of the handle. Nevertheless, despite of suboptimal matching, such an object model can be helpful when trying to grasp objects.

5.2.2. Segmentation Quality

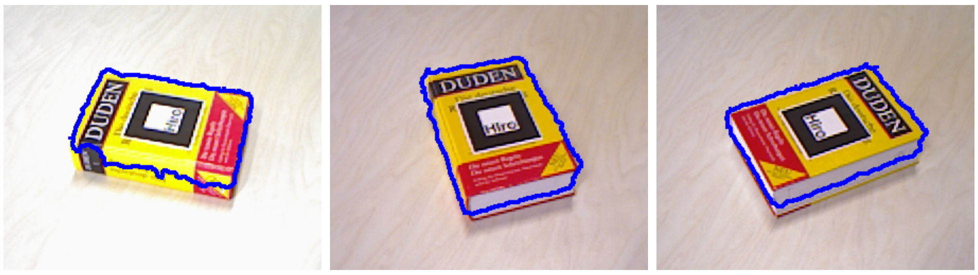
We project the object model into the color image to do a segmentation in each view. This experiment shows how the segmentation of an object evolves over time and analyzes the matching between ground truth and our segmentation.

We recorded the point clouds and color images of the scene over a number of frames. Each color image was labeled by hand into background and object. Then, we projected the object model generated by our approach into the color image to retrieve the segmentation.

Figure 5.7 contains two examples for the segmentation. The first image in each case is the segmentations after the first manipulations. In the beginning, the areas at the edge of the object are missing in the segmentation. After each manipulation the segmentation becomes better. In the third image of the shoe we can see one of the problems of such a segmentation. Despite the object was seen several times in this direction, the segmentation cuts off some area of the object at the bottom. This is caused by holes in the point cloud. Since the sensor was mounted above the table at a 45° angle, the sensor is not able to compute the depth information on the concavity of the



(a) Segmentation of a shoe after the views 1, 12 and 21



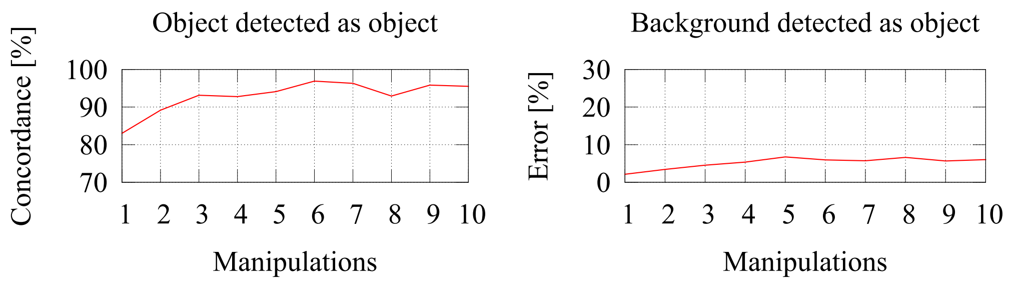
(b) Segmentation of a book after the views 1, 6 and 9

Figure 5.7.: We use a projection of the object model into the color image to do the segmentation. The two examples show the segmentation after a different number of manipulations.

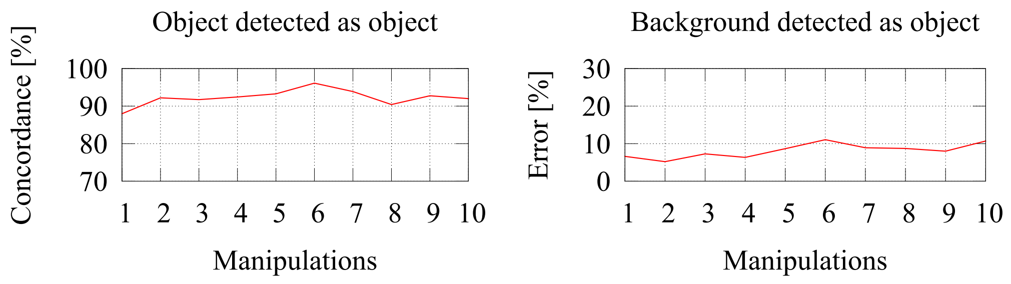
shoe. Thus, no points are available in this area, which results in holes in the segmentation, too.

Figure 5.8 shows how the segmentation accuracy evolves over the number of manipulations. The plots on the left side in Figure 5.8 show the number of pixels, which are correctly detected as belonging to the object in the current view (ground truth was labeled by hand). The correspondence between segmentation and ground truth reaches 95% after six manipulations. As mentioned above, the point clouds can contain holes due to the mounting angle of the sensor. Thus, it may happen that it cannot sense concavities or areas that have normal nearly perpendicular to the viewing direction of the sensor. This is the reason why the curves in Figure 5.8 can decrease, e.g., when the current view contains more of such points shadowed from the sensor's field of view.

Additionally, the plots on the right side in Figure 5.8 show the number of pixels that belong to the background and were wrongly labeled as object.



(a) Shoe



(b) Book

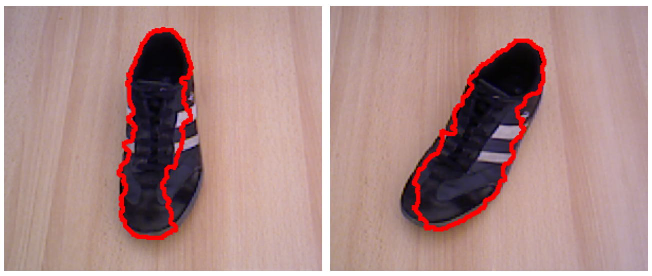
Figure 5.8.: The plots show the accuracy of the segmentation. It is given in percent of pixels of the ground truth segmentation.

The curve is rising due to ICP errors that accumulate over time.

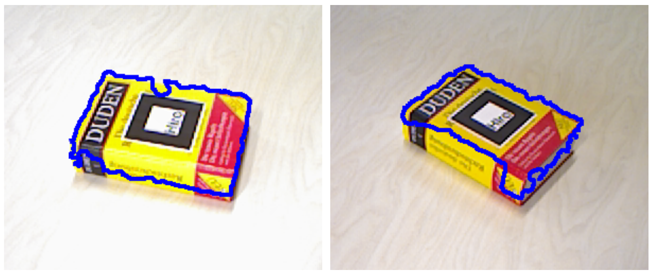
The segmentation has a high accuracy if the RGB-D sensor scans the scene without holes. Figure 5.7 (b) shows a book, where the point clouds does not contain holes. The segmentation is almost complete. The remaining missing areas are due to implementation issues. For getting the change sets we need to use a threshold to compare the depths. This threshold yields the missing segmentation area.

Furthermore, we can project the change sets only after each manipulation. This yields a very accurate partial segmentation, since the change sets consist of points that belong to the object. Figure 5.9 (a) and (b) displays the partial segmentations for two objects. The mean errors of correctly labeled pixels of the partial segmentations for the shoe and book are 97.8% and 95.4%, respectively.

Our segmentation could be used to initialize a color image based segmentation algorithm as a post-processing step to improve the accuracy.



(a) Shoe



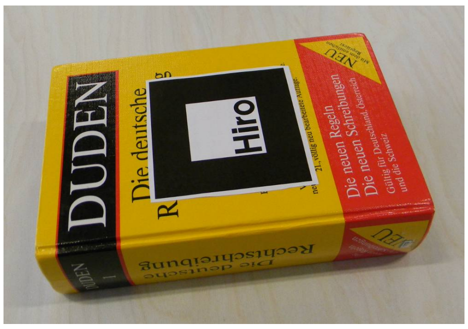
(b) Book

Figure 5.9.: If we project the change sets only, we get very accurate partial segmentations, since the change sets contain only points that belong to the object. These partial segmentations could be also used for training an object classifier.

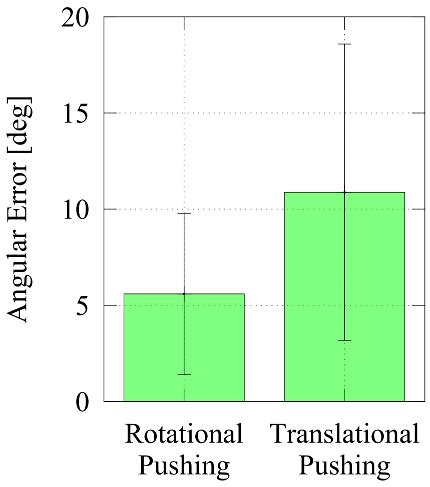
5.2.3. Accuracy of the Manipulations

We use pushing for manipulating objects. We propose to split the manipulation into rotational and translational pushing operations. This experiment shows the resulting accuracy of our push operations. The results base on experiments that were done with five different objects and were repeated twenty times per object (the object model improves over time, thus it was not reset after each manipulation). We examine the mean error and standard deviation of the manipulations. We compute the error by taking the difference between planned and executed manipulation. To assess the real motion of an object, we used an AR-marker that was attached to the center of the object and parallel to the table plane. Figure 5.10 (a) shows an object with a marker on it.

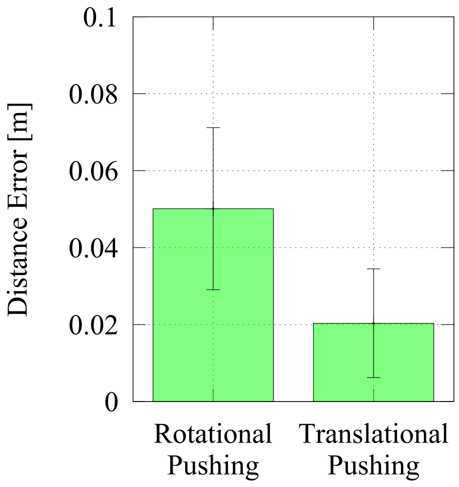
The Figures 5.10 (b) and (c) display the resulting errors and the corresponding standard deviations for rotational and translational push actions.



(a) AR-marker on an object



(b) Rotational error



(c) Translational error

Figure 5.10.: We used AR-markers in the experiments to estimate the real motion of an object. The plots show the mean errors and standard deviations of the push actions.

Figure 5.10 (b) shows the rotational errors, while 5.10 (c) contains the translational errors for all action types.

The mean angular error of the rotational manipulations is approximately $5.5°$ with a standard deviation $\sigma = 4.5°$. The translational error of the rotational manipulations is 0.05 m. We assumed that the center of rotation is equal to the geometric center (see Section 4.3.1 for more details). This approximating assumption causes the translation of the object.

The translational manipulations have an average error of 0.02 m. Although no rotation should take place, we get a mean rotational error of $11°$. This

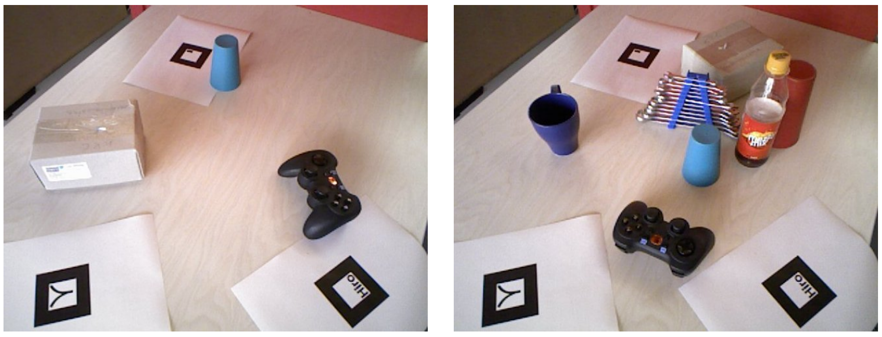

(a) Scene with three objects *(b)* Scene with seven objects

Figure 5.11.: A typical scene containing a number of objects. We need to estimate the working surface. By performing the RANSAC estimation with different numbers of objects we analyzed the influence on the estimation. The AR-marker were used to estimate the real table plane.

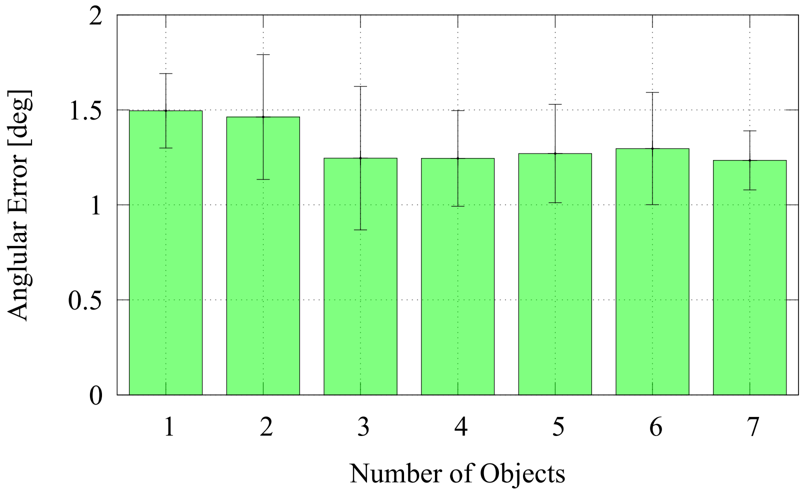
error results from the partial object model. A partial model may result in planning a suboptimal push action, since parts of the object are missing from the model so far.

Our rotational manipulations are currently not well suited for round objects, like cups. We are not able to move the gripper on a circular path around the geometric center to cause the rotation. Such objects require further work, e.g., performing the rotational manipulations by pushing the handle.

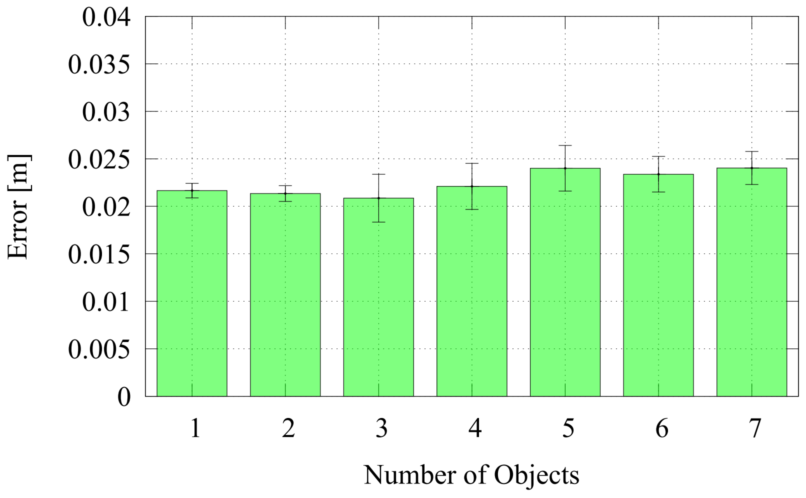
In conclusion, the pushing actions work for many objects. The physical assumptions have influence on the accuracy. Nonetheless, for modeling and segmentation the executed manipulations yield the necessary motion to model the whole object.

5.2.4. Table Plane Estimation

We also examined the influence of the number of objects on the RANSAC estimation of the planar surface, i.e., the table. Therefore, we placed three AR-markers on the table, which are used to determine the real table plane orientation. Figure 5.11 illustrates the setup. The data was generated with an increasing number of objects on the table. We rearranged the objects ten



(a) Angular error



(b) Distance error

Figure 5.12.: The plot shows, how the number of objects on the working surface influences the plane estimation (RANSAC). Therefore, we compare the RANSAC plane normal and the normal that was determined using three AR-markers. The experiment was repeated ten times for each number of objects. We rearranged the objects after each plane estimation.

times for each number.

We compared the plane normal from RANSAC with the normal that can be computed based on the locations of these three markers. Furthermore, we computed the mean distance at each marker position to the plane estimated by RANSAC. Figure 5.12 shows the resulting plots. In Figure 5.12 (a) we plot the mean angular error and its standard deviation between RANSAC estimation and the AR-marker orientation. We retrieve a low error of approximately 1.35° for the tested number of objects. The second plot in Figure 5.12 (a) displays the mean distance error between the marker positions and their orthogonal projections onto the RANSAC plane. The mean distance error is around 0.02 m. This error is mainly caused by the ARToolkit [3]. In conclusion, the experiments show that our RANSAC-based working surface estimation is suitable for a practical number of objects.

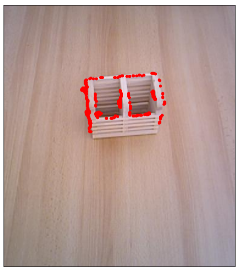
5.2.5. Filtering Point Clouds

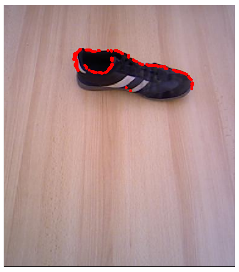
As mentioned in Section 3.3, there are points in the point cloud that vary in depth between consecutive frames. We are interested in removing these points because they would cause the change sets to contain points of other, not moved objects. This experiment analyzes, where these points appear and what threshold we can use to filter them out. Therefore, always ten depth images were recorded in a row. Then, we computed the standard deviation at each pixel over these ten depth images. The Figures 5.13 (a) - (d) show different objects. In each figure, we have marked the points with a standard deviation higher than a threshold $\epsilon_{\text{std}} = 0.005\,\text{m}$. They occur at the edges of objects because there points can jump between different depth levels due to sensor noise.

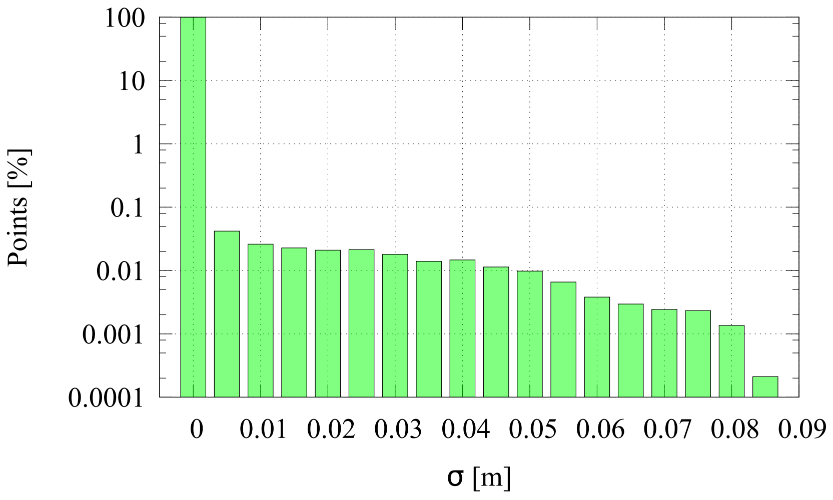
Figure 5.13 (e) shows the distribution of the standard deviations in a histogram. It was generated from various object views of four objects. We can see that most of the points (nearly 100%) have a standard deviation of zero or close to zero. Thus, using a threshold of $\epsilon_{\text{std}} = 0.005\,\text{m}$ yields the marked points in the Figures 5.13 (a) - (d).

(a) Cereal box

(b) Beaker


(c) Cutlery box


(d) Shoe



(e) Histogram of different standard deviations σ

Figure 5.13.: Noisy point cloud points: The red dots in the upper four figures correspond to points, which have an above-average standard deviation in depth. These points are neglected in further computation steps to avoid false positives in the change set of two consecutive point clouds. The histogram shows how often such points occur in point clouds. We filter them out using a threshold of $\epsilon_{\text{std}} = 0.005\,\text{m}$.

5.2.6. Parameters

There are three key parameters in the approach proposed in this thesis. The first one is the preferred angle in the planning step of the next manipulation. Setting this parameter too high may result in strange behavior of ICP (due to the large angle ICP may prefer another but wrong transformation). A too low number results in high execution times due to a large number of rotations required to model the full object. By experience, angles between 30° - 45° are the best ones to allow accurate ICP transformations and a relatively low execution time.

ξ_1 and ξ_2 were used to set the pushing distance in the corresponding object extents. A value of $\xi_1 = 0.5$ and $\xi_2 = 0.5$ are suited for many objects. Due to a bounded working area it is not recommendable to use higher values. Low values may result in insufficiently small movements, for which ICP has problems in matching the change sets.

Summary

With a series of experiments we evaluated the approach. The object models are accurate as long as ICP can match the change set with the view's point cloud. For rotationally symmetric objects there might be problems with ICP, e.g., the handle of a cup is wrongly matched. The segmentation of the color images is based on the object models. Therefore, the quality of the segmentation depends on the quality of the object model. We found out that the segmentation corresponds to the ground truth segmentation if the mounting angle of the RGB-D sensor allows for depth measurements for all areas of the object. Otherwise, there can be areas of the object that are missed by the segmentation. The overall quality of the object models and the segmentation of the color images are well suited for many meaningful tasks in robotics.

The manipulation is done by pushing. Therefore, we used the manipulator and a gripper with two fingers to allow for stable pushing operations. We divided the manipulations in rotational and translational pushing operations. The rotational manipulation always causes a translation of the object, too, because we assume that the center of rotation is equal to the geometric center. On the other hand, the translational manipulations can also cause a rotation, since planning is done with a partial object model.

Our pushing and planning algorithm require a planar working surface since it is done in 2D. We examined the influence of the number of objects on the estimation of the plane. It turned out, that a suitable number of objects has no influence on the accuracy of the plane estimation. Finally, we evaluated the filtering process. It is needed to remove points at object edges, which jump between different levels in the point cloud. Hence, they are spuriously part of the change sets of other objects. The points can be filtered out by checking the standard deviation over a number of point clouds at each point.

6 Conclusions

In this thesis we propose an approach that is used for autonomous exploration of objects in the environment. We present solutions for three key problems: First, we present an algorithm that iteratively models and segments objects based on their motion in the point clouds provided by a RGB-D sensor. Second, we show how a robotic manipulator can be used to move the objects in the scene. And third, we propose an algorithm for combined object modeling and manipulation to achieve a full three-dimensional object model and segmentation for each movable object in the reachable environment. The extracted 3D object models can be used for, e.g., grasping, while the segmented color image can be employed to train an object detection algorithm.

Moving an object in the scene creates changes in the point clouds, which we call change sets. Using these change sets we can decide, which part of the point cloud belongs to the moved object using a set of rules we introduced. Applications of the rules yields two change sets: one containing disappearing object points and one containing newly appearing object points. We use ICP to match the disappearing points against the current view to estimate the transformation of the object between two views. By combining the change sets using the estimated transformations we incrementally build a 3D model of the object. This procedure is advantageous, since we add only those points, which are certain to belong to the object. Thus, sensor readings of the background are reliably excluded from the model. Furthermore, we segment object and background in the color images by projecting the acquired object model or change sets into the image.

Besides object modeling the manipulation is an important component in the context of autonomous object exploration. We propose to manipulate object candidates by pushing. It allows interacting with objects that might be too heavy or large for grasping. The actual pushing operations require knowledge about the center of rotation and the friction distribution between object and working surface. Since both are not available in most practical applications, we suggest using the geometric center to approximate the center of rotation. This assumption is justified for many objects as has been shown in our experiments. For stable manipulations, we propose to use an end-effector equipped with two fingers. As there are two contact points between the end-effector and the object it prevents tilting of the object. Additionally, our approach divides the manipulation into rotational and translational push operations, which are executed separately.

We propose a framework that detects object candidates on a working surface and employs planning algorithms to incrementally generate 3D models and segmentations for the actually movable objects. It accounts for collisions with the scene and respects the limits of the manipulator workspace. Thereby, it ensures that models can be generated for the initially unknown objects employing the above methods.

In the experiments we evaluated the different aspects of our approach. We are able to retrieve accurate object models without any manual interference. On average, the segmentation reaches 95% accuracy after 6 manipulations. By projecting the change sets into the corresponding color image, we retrieve a partial segmentation of the object. The segmented area belongs with a probability of approximately 96% to the object. The geometric center has been confirmed to be a reasonable approximation of the center of rotation in our application. The resulting push operations are sufficiently accurate to explore the environment and allow robust pushing. Rotating an object causes a mean error of 5 degrees. The translational manipulation has a mean error of only 0.02 m.

In conclusion, our approach enables a robotic system to autonomously interact with initially unknown objects on a working surface to acquire accurate 3D models and color image segmentations suitable for classifier training.

Future Work

We had some problems with the bounded working area of our manipulator. There are two possible solutions to overcome these limitations. On the one hand, the manipulator can be mounted in another way, e.g., above the table to reach more objects. On the other hand, we could use the robotic platform to move the manipulator around the objects.

Additionally, learning an object model takes some time if we use only one RGB-D sensor. Relaxing the assumption of a statically mounted RGB-D sensor, the modeling can be performed emulating several statically mounted sensors at different view points. By mounting the sensor on the end-effector of the manipulator we are capable of moving the camera to these view points distributed around the object. Using the inverse kinematics or a RGBD-SLAM based approach we can compute the transformations between each sensor pose. Thus, we retrieve point clouds of the object from different view points. After a single manipulation of the object the new data is collected from all sensor view points. Computing the change sets in each sensor view would allow building a (full) 3D model without ICP. In addition to the reduced number of interactions necessary, the segmentations would be more accurate compared to the ICP-based model.

$\mathcal{A}$

Mathematical Background

A.1. Eigenvectors and Eigenvalues

We introduce *eigenvectors* and *eigenvalues* for a better understanding of the related *singular value decomposition*, which will be explained in the following section. An eigenvector $\vec{\mathbf{x}} \in \mathbb{R}^n$ of a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a vector that differs after multiplication with $\mathbf{A}$ only by a multiplicative scalar $\lambda \in \mathbb{R}$:

$$\mathbf{A} \cdot \vec{\mathbf{x}} = \lambda \cdot \vec{\mathbf{x}}. \tag{A.1}$$

This scalar is the corresponding *eigenvalue* λ. It is obvious that all vectors that have the same direction as the eigenvector are also eigenvectors. Only the zero vector is not considered to be an eigenvector.

Equation (A.1) can only be fulfilled iff

$$\det(\mathbf{A} - \lambda \mathbf{I}) = 0, \tag{A.2}$$

where $\det(\cdot)$ is the determinant of a matrix and $\mathbf{I} \in \mathbb{R}^{n \times n}$ the identity matrix. The eigenvalues of $\mathbf{A}$ are the roots of the characteristic polynomial

$$\chi_{\mathbf{A}}(\lambda) = \det(\mathbf{A} - \lambda \mathbf{I}). \tag{A.3}$$

After expanding Equation (A.3) we get a n-th degree polynomial in λ. Thus, there are n eigenvalues, which have not to be distinct. Depending on $\chi_{\mathbf{A}}(\lambda)$ there can be eigenvalues corresponding to the same root.

Finally, the eigenvectors are computed by solving a system of linear equations:

$$(\mathbf{A} - \lambda \mathbf{I}) \cdot \vec{\mathbf{x}} = 0. \tag{A.4}$$

Solving the eigenvalue problem is difficult for high dimensions. There are various algorithms to solve the problem numerically including the *QR algorithm* and the *Jacobi eigenvalue method*. Further details can be found in [36].

A.2. Singular Value Decomposition

The *singular value decomposition* (SVD) [7] is a widely used technique. Example applications are image processing, statistics or physics. It is a decomposition of a matrix in three matrices – two unitary matrices containing the singular vectors and a diagonal matrix containing the singular values. We use the SVD for solving the principle component analysis, which in turn is used to compute an extent estimation of an object (see Section A.3).

The SVD of a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ of rank r can be written as

$$\mathbf{A} = \mathbf{U \Sigma V}^*, \tag{A.5}$$

where $\mathbf{V}^*$ is the conjugate transpose of $\mathbf{V}$ and $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ is a diagonal matrix

$$\mathbf{\Sigma} = \mathrm{diag}(\sigma_1, \sigma_2, ..., \sigma_r), \quad \sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r \geq 0, \tag{A.6}$$

composed of the singular values σ_i of $\mathbf{A}$. $\mathbf{U} \in \mathbb{R}^{m \times m}$ and $\mathbf{V} \in \mathbb{R}^{n \times n}$ are unitary matrices that contain the left and right singular vectors of $\mathbf{A}$:

$$\mathbf{U} = (\mathbf{u}_1, ..., \mathbf{u}_m), \quad \mathbf{V} = (\mathbf{v}_1, ..., \mathbf{v}_n). \tag{A.7}$$

The relationship to the eigenvalue decomposition is as follows. The columns of $\mathbf{U}$ and $\mathbf{V}$ are the eigenvectors of $\mathbf{AA}^*$ and $\mathbf{A}^*\mathbf{A}$, respectively. Furthermore, the singular values σ_i are the positive, nonzero eigenvalues of $\mathbf{AA}^*$ and $\mathbf{A}^*\mathbf{A}$.

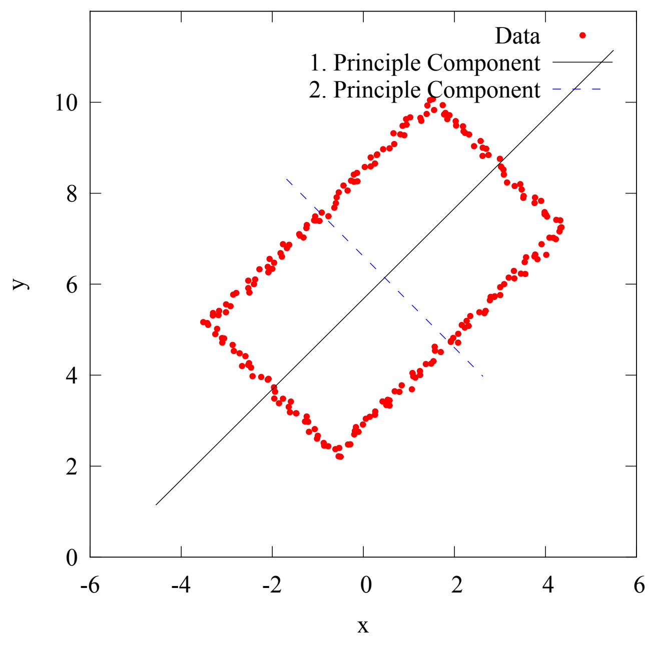


Figure A.1.: The two principle components of an object are used to determine the extent of an object.

A.3. Principal Component Analysis

The *principle component analysis* (PCA) [24] converts a set of observations into linearly uncorrelated values (*principle components*) using an orthogonal transformation. Typically, a data set consists of n elements with p measured properties. Hence, there are n elements in a p-dimensional space. The main aim of PCA is to reduce the number of measurements to a sub space $\mathbb{R}^q$ with $q \leq p$ without losing too much information. Each of the principle components is orthogonal to the others. The first principle component corresponds to the direction of the largest variance. PCA can be implemented as an eigenvalue decomposition of the data covariance matrix or as a singular value decomposition of the data matrix.

In this thesis, we use it to estimate the two axes with the largest variance of a two-dimensional object that is represented by a set of points. Projecting all points on the axes allows roughly computing the dimensions of an object. Figure A.1 shows the two principle components of an object given by a set of points. The estimated extent of an object is needed when planning push operations to manipulate it (see Chapter 4).

$\mathcal{B}$

Algorithmic Basics

B.1. Normal Vector Estimation

The normal vector is an important property of a surface. We need to know about normals to compute push operations. How an object behaves when it is pushed at some location depends, amongst others, on its surface normal. We can approximate the problem of normal estimation by a tangential plane on the surface. The computation of such a plane is a least-square plane fitting problem in the neighborhood of the query point.

Let the neighborhood of a point $\mathbf{p}_k$ be the set $\mathcal{P}_k$, $|\mathcal{P}_k| = n$. The neighborhood can be computed via *kd-trees* or *octrees* [16]. Estimating the plane is done in the least-square sense. Rusu [40] proposes to take the arithmetic mean of all points in $\mathcal{P}_k$

$$\mathbf{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} \mathbf{p}_i \tag{B.1}$$

as the position $\mathbf{x}$ and setting up the covariance matrix $\mathbf{C} \in \mathbb{R}^{3\times3}$ as follows:

$$\mathbf{C} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{p}-\mathbf{x})(\mathbf{p}-\mathbf{x})^T, \quad \mathbf{C}\cdot\vec{\mathbf{v}}_j = \lambda_j \cdot \vec{\mathbf{v}}_j, \quad j \in \{1,2,3\}. \tag{B.2}$$

Then, the consideration of the eigenvectors $\vec{\mathbf{v}}_j$ and eigenvalues λ_j of $\mathbf{C}$ allows the computation of $\vec{\mathbf{n}}$. The normal $\pm\vec{\mathbf{n}}$ is given by the eigenvector corresponding to the smallest eigenvalue.

The sign of the direction cannot be uniquely determined. Therefore, the direction of the normal must be flipped if it points away from the camera's point of view $\mathbf{c}$. Every normal must satisfy the condition

$$\vec{\mathbf{n}}_i \cdot (\mathbf{c} - \mathbf{p}_i) > 0. \tag{B.3}$$

Otherwise, we have to take the inverse direction:

$$\vec{\mathbf{n}}_i \leftarrow -\vec{\mathbf{n}}_i. \tag{B.4}$$

B.2. RANSAC

The *RANSAC* algorithm (**ran**dom **sa**mple **c**onsensus) is used to estimate model parameters. Example applications of the algorithm are "feature-matching" (e.g., the combination of single images to a big panorama image [8]) or the recognition of geometrical primitives as planes or spheres. It was introduced by Fischler and Bolles [15]. We use RANSAC to estimate the surface plane on which the objects are placed and manipulated.

Let $\mathcal{D} = \{\mathbf{x}_i \,|\, 1 \leq i \leq n\}$ be a set of observations. Each observation is a vector $\mathbf{x}_i \in \mathbb{R}^d$ in the d-dimensional space. Depending on what we want to do with the algorithm we have to define an appropriate model M. Since we want to detect a planar surface we choose

$$M : ax + by + cz + d = 0, \quad (x, y, z) \in \mathbb{R}^3, \quad a, b, c, d \in \mathbb{R}, \tag{B.5}$$

which describes a two-dimensional plane in the 3D space. The vector θ contains the minimal number of parameters that are necessary for a unique definition of M. Using the model in Equation (B.5), we have a four dimensional parameter vector $\theta = (a, b, c, d) \in \mathbb{R}^4$.

In the first step the algorithm computes a subset $\mathcal{D}_{\text{min}} \subset \mathcal{D}$, which is sufficient to describe M. $\mathcal{D}_{\text{min}}$ is chosen in a random and uniform way. Thus, θ is given by

$$\theta = f_M(\mathcal{D}_{\text{min}}), \tag{B.6}$$

where $f_M(\cdot)$ computes the corresponding parameter vector θ based on the minimal subset $\mathcal{D}_{\text{min}}$ (i.e., it fits a plane through three points).

Next, the quality of θ is analyzed on $\mathcal{D}$ using an error function $e(\cdot)$. We define the error function as follows:

$$e_M(\mathbf{x}, \theta) = \text{dist}(\mathbf{x}, M_\theta). \tag{B.7}$$

dist$(\cdot)$ is a distance function which computes the distance between an observation $\mathbf{x}$ and the concrete model M_θ. In the context of plane fitting the orthogonal distance between $\mathbf{x}$ and the plane given by M_θ can be used.

Now, the set of inliers is computed by

$$\mathcal{I}_{M_\theta} = \{x \in \mathcal{D} \,|\, e(\mathbf{x}, M_\theta) \leq \epsilon\}\,, \tag{B.8}$$

where ϵ gives the maximum allowed error. Inliers are observations that are conform with θ in M. Regarding the plane estimation, $\mathcal{I}_{M_\theta}$ contains the points, which are close to the plane defined by θ.

By repeating this process, we get a set of possible parameters and inliers, which are in accordance with the model. Finally, we have to decide, which parameter vector to use. We choose the parameter vector $\hat{\theta}$ that has the most inliers. Thus, we get

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}(|\mathcal{I}_{M_\theta}|) \tag{B.9}$$

as the final parameter vector.

An important parameter of the RANSAC algorithm is the number of iterations. If the number is too low the resulting concrete model M_θ will be of poor quality. If we use more iterations the runtime will increase.

More information about the algorithm can be found in the original paper [15] or in [47].

B.3. Potential Fields

Potential fields are often used in planning. Theoretically, a potential field is a n-dimensional grid containing a potential, i.e., a scalar value in each grid cell. For example, the grid can be two dimensional where each grid cell is the location on a floor of a room. Obstacles and walls can then be modeled using the potential values. A low value corresponds to free space, while a high value stands for an obstacle. If we search a path between two points, we can use the potential field as costs for path planning.

Mathematically, we can regard a potential [46, 49] as a function of the type

$$U : \mathcal{C} \to \mathbb{R}, \tag{B.10}$$

where $\mathcal{C}$ is the configuration space. A low potential represents an attractive potential (Figure B.1 (a)), while a high potential is repulsive (Figure B.1 (b)). To retrieve the final potential of each configuration both the attractive and repulsive potentials are summed up:

$$U(\mathbf{c}) = U_+(\mathbf{c}) + U_-(\mathbf{c}), \tag{B.11}$$

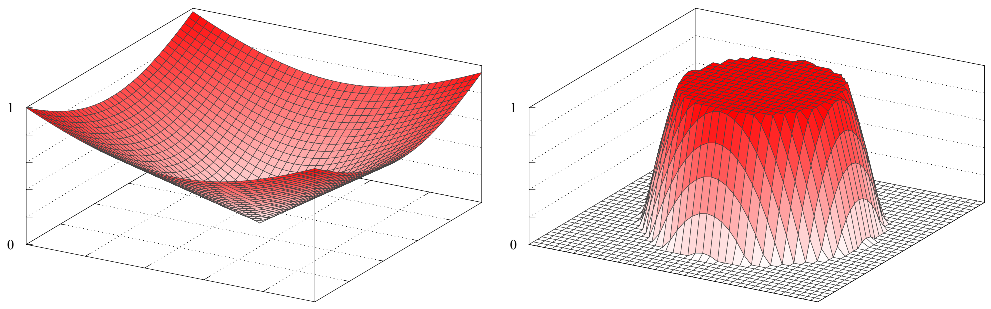
where $\mathbf{c} \in \mathcal{C}$ is a configuration.

Basically, an attractive potential implies free space, and thus configurations that are preferable. We choose the new goal configuration as the configuration with the minimum potential value. Then, we use the potential values as a cost function to find a way from the current configuration to the configuration with the lowest potential.

Typically, a force is computed at each configuration to find a path through the configuration space. A force

$$F(\mathbf{c}) = -\nabla U(\mathbf{c}) \tag{B.12}$$

is the gradient of the differentiable function $U(\mathbf{c})$. Therefore, the path planning is based on a gradient descent method and stops if a minimum is



(a) Example for an attractive potential *(b)* Example for a repulsive potential

Figure B.1.: Attractive and repulsive potentials: They are used to represent the configuration space of an object.

reached. Since this method can stick to a local minimum the A* search algorithm can be applied to find a global solution.

In this thesis, we use the potential fields for planning the object motion on the working surface. More specific, the potential field represents the configuration space of an object, i.e., it contains all reachable transformations of it on a plane and assign a potential value to each to them. Finally, a path planning algorithm is used to find a way from the current object's pose to the pose with the lowest potential value.

B.4. The A* Algorithm

The *A* algorithm* [39] computes a path between two nodes in a graph. We use A* for moving an object from one location to another in a collision-free way. It uses a heuristic $h(\cdot)$ to estimate the costs from the current search node to the goal node. A* is complete (it finds a solution if one exists) and optimal if an admissible heuristic is used. A heuristic is admissible if

$$h(x) \leq d(x, y) + h(y), \tag{B.13}$$

where $d(x, y)$ corresponds to real costs between node x and y. This means, that the heuristic never overestimates the costs to the goal node. $h(\cdot)$ either matches with the real cost function or underestimates it. An example for an

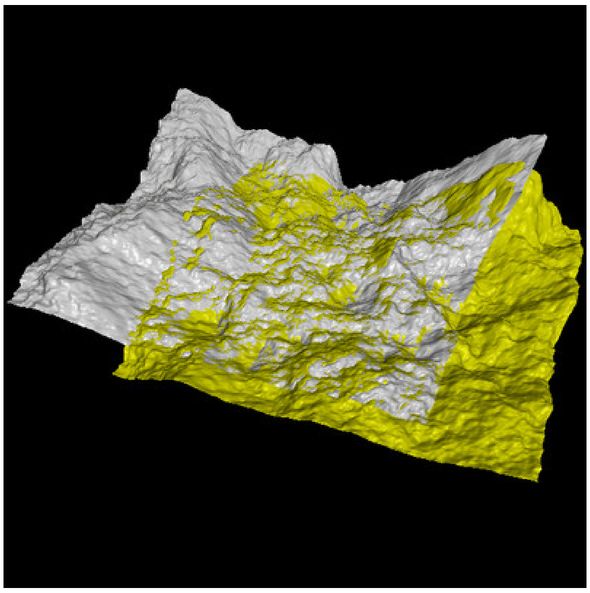


Figure B.2.: The iterative closest point algorithm is used to align two point clouds [1].

admissible heuristic is the Manhattan distance in the context of finding a way through a maze, which is represented by a grid.

B.5. Iterative Closest Point

The *iterative closest point* algorithm (ICP) [52] is employed for aligning two point clouds (see Figure B.2 for an example). We use it to find the transformation between two object poses by fitting the change set against the current point cloud (see Section 3.2 for more details).

Given two point clouds $\mathcal{X}$ and $\mathcal{P}$ and a threshold ϵ, which gives the maximum error between both point clouds, we want to find a rotation $\mathbf{R} \in \mathbb{R}^{3\times3}$ and a translation vector $\vec{\mathbf{t}} \in \mathbb{R}^3$ to minimize the error function

$$\mathrm{E}(\mathbf{R}, \vec{\mathbf{t}}) = \frac{1}{|\mathcal{P}|} \sum_{i=1}^{|\mathcal{P}|} \left\| \mathbf{x}_i - \left(\mathbf{R}\mathbf{p}_i + \vec{\mathbf{t}} \right) \right\|_2^2, \tag{B.14}$$

where $\mathbf{x}_i$ and $\mathbf{p}_i$ are corresponding points.

The algorithm consists of five steps:

1. Associate corresponding points between $\mathcal{X}$ and $\mathcal{P}$. The simplest association is based on nearest neighbors. Another method for correspondence estimation is normal shooting, where a projection along the normal is used to find the intersection with the other point cloud.
2. Estimate the rotation $\mathbf{R}$ and translation $\vec{\mathbf{t}}$ between $\mathcal{X}$ and $\mathcal{P}$ using the point correspondences. The estimation can be done in closed form using a SVD [5].
3. Apply the estimated transformation to $\mathcal{X}$.
4. Compute the error $\mathrm{E}(\mathbf{R}, \vec{\mathbf{t}})$ between the transformed point cloud $\mathcal{X}$ and $\mathcal{P}$.
5. If $\mathrm{E}(\mathbf{R}, \vec{\mathbf{t}})$ was decreased and $\mathrm{E}(\mathbf{R}, \vec{\mathbf{t}}) > \epsilon$ iterate the above steps. Otherwise the final transformation is found.

B.6. Concave Hull Estimation

A *concave hull* is similar to a convex hull with the difference that it also can model concavities. It can be regarded as the estimation of the shape of an object. The hull borders the data more tightly. In contrast to the convex hull, the concave hull is not unique.

We use the concave hull to find positions to push the object. Since we plan the manipulations on a 2D plane the concave hull is computed around the projected points on this plane. In general, a convex hull is not sufficient for many objects.

One way to estimate the concave hull consists of a *Delaunay triangulation* followed by a merging step of the triangles. The merging can be done with the *alpha shape* (or α shape) algorithm.

Let the data be given as a set of points $\mathcal{P} = \{\mathbf{p}_1, ..., \mathbf{p}_n\}$, where each point is a two dimensional point $\mathbf{p}_i \in \mathbb{R}^2$. The Delaunay triangulation computes a mesh of triangles, such that the circle that contains the three triangle points

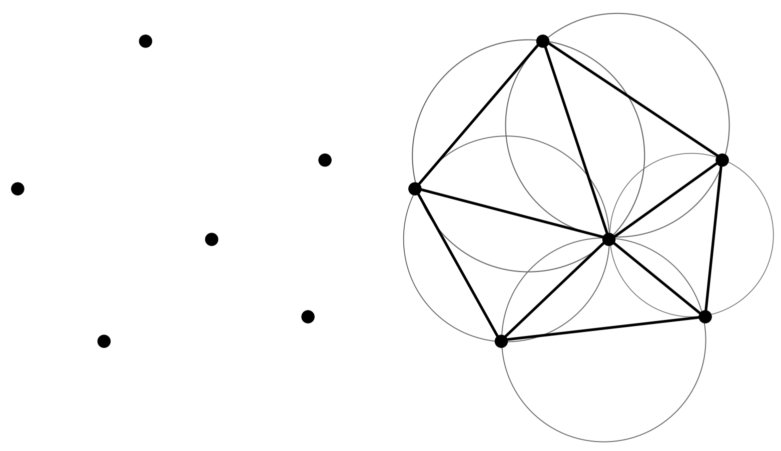

(a) Point set *(b)* Valid Delaunay triangulation

Figure B.3.: A valid Delaunay triangulation of the given point set. Only the three points of the corresponding triangle are located in the circle, which is defined by those three points.

does not contain any other points. More information can be found in [11]. An example is shown in Figure B.3.

Based on the Delaunay triangulation, the concave hull is computed using α-shapes [13, 14]. It is a generalization of the convex hull. The algorithm is based on a single parameter α with $0 \leq \alpha \leq \infty$. It controls the maximum curvature of a cavity. The shape is estimated using the triangles from the Delaunay triangulation. They are included in the α-shape if the corresponding circumsphere has a radius of at most α. For $\alpha = \infty$, the shape is equal to the convex hull. By using $\alpha = 0$ only the points of the point set belong to the α-shape. Using an appropriate value for α allows to estimate the desired shape of objects. The union of all triangles, which fulfill the above condition gives the final shape. See [14] for details.

Bibliography

[1] Iterative closest point algorithm. online, 2012. URL `http://ais.informatik.uni-freiburg.de/teaching/ss12/robotics/slides/17-icp.pdf`.

[2] Artoolkit home page. online, 2013. URL `http://www.hitl.washington.edu/artoolkit/`.

[3] Daniel F Abawi, Joachim Bienwald, and Ralf Dorner. Accuracy in optical tracking with fiducial markers: an accuracy function for artoolkit. In *Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 260–261. IEEE Computer Society, 2004.

[4] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(5):898–916, 2011.

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

[6] S. Awate, T. Tasdizen, and R. Whitaker. Unsupervised texture segmentation with nonparametric neighborhood statistics. *Computer Vision–ECCV 2006*, pages 494–507, 2006.

[7] Ake Bjõrck. *Numerical Methods for Least Squares Problems*. SIAM: Society for Industrial and Applied Mathematics, 1996. ISBN 0898713609.

[8] Matthew Brown and David G. Lowe. Automatic panoramic image stitching using invariant features. *International Journal of Computer Vision*, 74(1):59–73, 2007.

[9] T. Brox and J. Malik. Object segmentation by long term analysis of point trajectories. In *European Conference on Computer Vision (ECCV)*, Lecture Notes in Computer Science. Springer, Sept. 2010.

[10] D. Cremers and S. Soatto. Motion competition: A variational approach to piecewise parametric motion segmentation. *International Journal of Computer Vision*, 62(3):249–265, 2005.

[11] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2008. ISBN 9783540779735.

[12] M.R. Dogar and S.S. Srinivasa. Push-grasping with dexterous hands: Mechanics and a method. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2123–2130. IEEE, 2010.

[13] Herbert Edelsbrunner and Ernst P Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics (TOG)*, 13(1):43–72, 1994.

[14] Herbert Edelsbrunner, David Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. *Information Theory, IEEE Transactions on*, 29(4):551–559, 1983.

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

[16] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice in C (2nd Edition)*. Addison-Wesley Professional, 1995. ISBN 0201848406.

[17] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge books online. Cambridge University Press, 2004. ISBN 9780521540513.

[18] 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)*, volume 20, pages 22–25, 2010.

[19] Tucker Hermans, James M. Rehg, and Aaron F. Bobick. Guided pushing for object singulation. In *IROS*, pages 4783–4790, 2012.

[20] K. Hsiao, S. Chitta, M. Ciocarlie, and E.G. Jones. Contact-reactive grasping of objects with partial shape information. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 10, page 2010, 2010.

[21] ASUSTEK COMPUTER INC. Asus reveals the world's first motion-sensing experience for pc. online, 02 2011. URL `http://milandesignweek2011.asus.com/archives/cebit-2011/asus-reveals-the-worlds-first-motion-sensing-experience-for-pc`.

[22] ASUSTEK COMPUTER INC. Xtion pro live. online, 2011. URL `http://www.asus.com/Multimedia/Motion_Sensor/Xtion_PRO_LIVE`.

[23] 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.

[24] I.T. Jolliffe. *Principal Component Analysis*. Springer, 2002. ISBN 0387954422.

[25] Dov Katz, Moslem Kazemi, J. Andrew (Drew) Bagnell, and Anthony (Tony) Stentz. Clearing a pile of unknown objects using interactive perception. Technical report, Robotics Institute, Pittsburgh, PA, November 2012.

[26] J. Kim, J.W. Fisher III, A. Yezzi, M. Çetin, and A.S. Willsky. A nonparametric statistical method for image segmentation using information theory and curve evolution. *Image Processing, IEEE Transactions on*, 14(10):1486–1502, 2005.

[27] M. Krainin, B. Curless, and D. Fox. Autonomous generation of complete 3d object models using next best view manipulation planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5031–5037. IEEE, 2011.

[28] C. Li, C. Xu, C. Gui, and M.D. Fox. Level set evolution without reinitialization: A new variational formulation. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 430–436. IEEE, 2005.

[29] K.M. Lynch and M.T. Mason. Stable pushing: Mechanics, controllability, and planning. *The International Journal of Robotics Research*, 15(6): 533–556, 1996.

[30] M.T. Mason. *Manipulator grasping and pushing operations*. PhD thesis, Massachusetts Institute of Technology, 1982.

[31] M.T. Mason. *Mechanics of Robotic Manipulation*. Intelligent Robots and Autonomous Agents. MIT Press, 2001. ISBN 9780262133968.

[32] M.J. Mataric, M. Nilsson, and KT Simsarin. Cooperative multi-robot box-pushing. In *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, volume 3, pages 556–561. IEEE, 1995.

[33] P. Ochs and T. Brox. Object segmentation in video: a hierarchical variational approach for turning point trajectories into dense regions. In *IEEE International Conference on Computer Vision (ICCV)*, 2011.

[34] D. Omrcen, C. Boge, T. Asfour, A. Ude, and R. Dillmann. Autonomous acquisition of pushing actions to support object grasping with a humanoid robot. In *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*, pages 277–283. IEEE, 2009.

[35] M.A. Peshkin and A.C. Sanderson. The motion of a pushed, sliding workpiece. *Robotics and Automation, IEEE Journal of*, 4(6):569–598, 1988.

[36] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 2007. ISBN 0521880688.

[37] D. Rao, Q.V. Le, T. Phoka, M. Quigley, A. Sudsang, and A.Y. Ng. Grasping novel objects with depth segmentation. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2578–2585. IEEE, 2010.

[38] ROS. Ros wiki. online, 2013. URL `http://www.ros.org/wiki/`.

[39] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, 2010. ISBN 9780136042594.

[40] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Computer Science department, Technische Universität München, Germany, October 2009.

[41] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[42] A. Saxena, J. Driemeyer, and A.Y. Ng. Robotic grasping of novel objects using vision. *The International Journal of Robotics Research*, 27(2):157–173, 2008.

[43] Germany SCHUNK GmbH & Co. KG, Lauffen. 2-finger parallel gripper wsg. online, 2013. URL `http://www.us.schunk.com/schunk/schunk_websites/products/latest_products_detail.html?article_id=14322`.

[44] J. Shi and J. Malik. Motion segmentation and tracking using normalized cuts. In *Computer Vision, 1998. Sixth International Conference on*, pages 1154–1160. IEEE, 1998.

[45] J. Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

[46] R. Siegwart and I.R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Intelligent Robotics and Autonomous Agents. MIT Press, 2004. ISBN 9780262195027.

[47] P. H. S. Torr and A. Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78:2000, 2000.

[48] A. Wedel, T. Pock, C. Zach, H. Bischof, and D. Cremers. An improved algorithm for tv-l 1 optical flow. *Statistical and Geometrical Approaches to Visual Motion Analysis*, pages 23–45, 2009.

[49] T. Weigel, J.S. Gutmann, M. Dietl, A. Kleiner, and B. Nebel. Cs freiburg: coordinating robots for successful soccer playing. *Robotics and Automation, IEEE Transactions on*, 18(5):685–699, 2002.

[50] K.M. Wolter. *Introduction to Variance Estimation*. Statistics for social and behavioral sciences. Springer London, Limited, 2007. ISBN 9780387350998.

[51] L. Xu, J. Jia, and Y. Matsushita. Motion detail preserving optical flow estimation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(9):1744–1757, 2012.

[52] C. Yang and G. Medioni. Object modelling by registration of multiple range images. *Image and vision computing*, 10(3):145–155, 1992.

[53] K. Zhang, L. Zhang, H. Song, and W. Zhou. Active contours with selective local or global segmentation: A new formulation and level set method. *Image and Vision Computing*, 28(4):668–676, 2010.

List of Figures

1.1. Typical scene with objects 2
(a). Scene 2
(b). Reconstructed 3D model 2
1.2. KinectFusion
Source: [23] 7
1.3. Grasping an object after pushing
Source: [34] 9

2.1. Overview 12

3.1. Change set between two point clouds 18
(a). View before motion 18
(b). View after motion 18
3.2. Change set (new depth < old depth) 19
(a). Difference between old points 19
(b). Change set (known object in old view) 19
(c). Change set (unknown object in old view / initial situation) 19
3.3. Change set (new depth > old depth) 20
(a). New points 20
(b). Resulting change sets 20
3.4. Examples of change sets 20
(a). Example 1 20

(b). Example 2 20
3.5. Example of ICP matching 22
3.6. Filtering point clouds 24
3.7. 2D segmentation 26
3.8. Pinhole camera model 26
(a). Pinhole camera model 26
(b). Focal length f 26

4.1. Examples of a potential fields 31
4.2. Pushing distance 34
(a). Principle component analysis 34
(b). Potential field 34
4.3. Translational pushing 38
4.4. Translational pushing 39
(a). Non-stable 39
(b). Stable 39
4.5. Rotational pushing 40
(a). Rotational pushing 40
(b). Pushing point 40

5.1. Experimental Setup: omniRob from KUKA 44
(a). Experimental setup 44
(b). KUKA omniRob 44
5.2. Gripper and RGB-D sensor 45
(a). Schunk WSG 50
Source: [43] 45
(b). Depth sensor: *ASUS XtionPRO LIVE*
Source: [22] 45
5.3. User interface of our implementation 46
5.4. Objects used in the experiments 47
(a). Shoe 47
(b). Book 47
(c). Cutlery box 47
(d). Cereal box 47

(e). Cup 47
(f). Beaker 47
5.5. Result of object modeling 48
(a). Shoe 48
(b). Book 48
(c). Cutlery box 48
(d). Cereal box 48
5.6. Object modeling: Rotationally symmetric objects 49
(a). Cup 49
(b). Beaker 49
5.7. Object segmentation 50
(a). Shoe 50
(b). Book 50
5.8. Segmentation accuracy 51
(a). Shoe 51
(b). Book 51
5.9. Segmentation using change sets only 52
(a). Shoe 52
(b). Book 52
5.10. Accuracy of the Push Operations 53
(a). AR-marker on an object 53
(b). Rotational error 53
(c). Translational error 53
5.11. Typical scene 54
(a). Scene with three objects 54
(b). Scene with seven objects 54
5.12. Influence of the objects on planar surface estimation 55
(a). Angular error 55
(b). Distance error 55
5.13. Filtering the point cloud 57
(a). Cereal box 57
(b). Beaker 57
(c). Cutlery box 57

(d). Shoe 57
(e). Result 57

A.1. Principal component analysis 67

B.1. Examples for potentials 73
(a). Attractive potential 73
(b). Repulsive potential 73
B.2. Iterative closest point
Source: [1] 74
B.3. Delaunay triangulation 76
(a). Point set 76
(b). Delaunay triangulation 76