



UNIVERSITÄT  
KOBLENZ · LANDAU

Fachbereich 4: Informatik



# Probabilistic Object Recognition Using Extended Implicit Shape Models

Masterarbeit  
zur Erlangung des Grades  
MASTER OF SCIENCE  
im Studiengang Computervisualistik

vorgelegt von

Norman Link

**Betreuer:** Dipl.-Inform. Viktor Seib, Institut für Computervisualistik,  
Fachbereich Informatik, Universität Koblenz-Landau

**Erstgutachter:** Prof. Dr.-Ing. Dietrich Paulus, Institut für  
Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau

**Zweitgutachter:** Dipl.-Inform. Viktor Seib, Institut für  
Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau

Koblenz, im Februar 2014



## Kurzfassung

Objekterkennung ist ein gut erforschtes Gebiet bei bildbasiertem Rechnersehen und eine Vielzahl an Methoden wurden entwickelt. In letzter Zeit haben sich dabei Ansätze verbreitet, die auf dem Implicit Shape Model-Konzept basieren. Dabei werden Objekte zunächst in grundlegende visuelle Bestandteile aufgetrennt, die um örtliche Informationen erweitert werden. Das so generierte Objektmodell wird dann in der Objekterkennung genutzt, um unbekannte Objekte zu erkennen. Seit dem Aufkommen von erschwinglichen Tiefenkameras wie der Microsoft Kinect wurde jedoch die Objekterkennung mittels 3D-Punktwolken von zunehmender Bedeutung. Im Rahmen des Robotersehens in Innenräumen wird ein Verfahren entwickelt, welches auf vorhandenen Ansätze aufbaut und damit die Implicit Shape Model basierte Objekterkennung für die Verarbeitung von 3D-Punktwolken erweitert.

## Abstract

Object recognition is a well-investigated area in image-based computer vision and several methods have been developed. Approaches based on Implicit Shape Models have recently become popular for recognizing objects in 2D images, which separate objects into fundamental visual object parts and spatial relationships between the individual parts. This knowledge is then used to identify unknown object instances. However, since the emergence of affordable depth cameras like Microsoft Kinect, recognizing unknown objects in 3D point clouds has become an increasingly important task. In the context of indoor robot vision, an algorithm is developed that extends existing methods based on Implicit Shape Model approaches to the task of 3D object recognition.



## Erklärung

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

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

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden. ja  nein

Der Veröffentlichung dieser Arbeit im Internet stimme ich zu. ja  nein

Koblenz, den 6. Februar 2014



# Contents

<b>1</b>	<b>Introduction</b>	<b>15</b>
1.1	Problem Statement . . . . .	16
1.2	Motivation . . . . .	17
1.3	Goal . . . . .	18
1.4	Implementation . . . . .	19
1.5	Outline . . . . .	19
<b>2</b>	<b>State of the Art</b>	<b>21</b>
2.1	Implicit Shape Model . . . . .	21
2.1.1	Bag-Of-Words . . . . .	21
2.1.2	Object Recognition . . . . .	22
2.2	Features . . . . .	27
2.2.1	Normals . . . . .	28
2.2.2	Keypoints . . . . .	29
2.2.3	Local Reference Frame . . . . .	32
2.2.4	Descriptors . . . . .	34
2.3	Mean-Shift Mode Estimation . . . . .	38
<b>3</b>	<b>Creating Implicit 3D Representations</b>	<b>43</b>
3.1	Overview . . . . .	43
3.2	Preprocessing . . . . .	45
3.2.1	Input Acquisition . . . . .	46
3.3	Features . . . . .	48
3.4	Clustering and Codebook Generation . . . . .	50
3.5	Activation . . . . .	54
<b>4</b>	<b>Probabilistic Object Recognition</b>	<b>61</b>
4.1	Features . . . . .	62
4.2	Activation . . . . .	63
4.3	Voting . . . . .	64
4.3.1	Weighting . . . . .	65

4.3.2	Rotation Invariance . . . . .	67
4.3.3	Hough-Voting . . . . .	70
4.3.4	Mean-Shift Mode Estimation . . . . .	71
4.4	Multi-Class Detection . . . . .	73
4.5	Bounding Box . . . . .	73
<b>5</b>	<b>Evaluation</b>	<b>77</b>
5.1	Datasets . . . . .	77
5.1.1	Kinect Dataset . . . . .	77
5.1.2	Stanford Dataset . . . . .	81
5.2	Parameter Selection . . . . .	82
5.2.1	Features . . . . .	83
5.2.2	Codebook Creation . . . . .	86
5.3	Classification . . . . .	90
5.4	Object Recognition . . . . .	93
5.4.1	Kinect Dataset . . . . .	93
5.4.2	Stanford Dataset . . . . .	97
5.5	Performance . . . . .	101
5.6	Discussion . . . . .	102
<b>6</b>	<b>Conclusion</b>	<b>105</b>
6.1	Further Prospects . . . . .	106
<b>A</b>	<b>Appendix</b>	<b>109</b>
A.1	Framework Overview . . . . .	109
A.2	Parameters . . . . .	110
A.2.1	General . . . . .	112
A.2.2	Keypoints . . . . .	113
A.2.3	Features . . . . .	114
A.2.4	Clustering . . . . .	116
A.2.5	Codebook . . . . .	118
A.2.6	Activation Strategy . . . . .	118
A.2.7	Voting . . . . .	119
A.3	Training-GUI . . . . .	121
A.4	External Components . . . . .	122



# List of Tables

2.1	Common profile and kernel types . . . . .	40
4.1	Detection process and bounding box extraction . . . . .	74
5.1	Training models from Kinect dataset . . . . .	79
5.2	Sample scenes from the Kinect dataset with their corresponding classes . . . . .	80
5.3	Training models from the Stanford dataset . . . . .	82
5.4	Sample scenes from the Stanford dataset with their corresponding classes . . . . .	83
5.5	Minimum distance between ground truth and detected objects . . . . .	84
5.6	Index of the minimum distance hypothesis . . . . .	85
5.7	Distance between ground truth and first object hypothesis . . . . .	85
5.8	Average results on the minimum distance object hypothesis for training case A . . . . .	94
5.9	Average results on the first-ranked object hypothesis for training case A . . . . .	94
5.10	Average results on the minimum distance object hypothesis for training case B . . . . .	95
5.11	Average results on the first-ranked object hypothesis for training case B . . . . .	96
5.12	Results from the Stanford dataset . . . . .	100
5.13	System specifications for the evaluation system . . . . .	101
5.14	Summary of parameters . . . . .	102



# List of Figures

1.1	The service robot LISA . . . . .	18
2.1	Recognition procedure from [LS03] . . . . .	24
2.2	Recognition procedure from [KPW <sup>+</sup> 10] . . . . .	26
2.3	Support sphere of the SHOT descriptor . . . . .	35
2.4	Illustration of point pairs within a local neighborhood used for PFH computation . . . . .	36
2.5	Illustration of point pair influences used for FPFH computation . . . . .	38
3.1	Training pipeline . . . . .	44
3.2	The Kinect camera . . . . .	47
3.3	Activation procedure during training . . . . .	58
4.1	Detection pipeline . . . . .	62
4.2	Activation procedure during detection . . . . .	69
5.1	Ground truth annotation on a point cloud . . . . .	81
5.2	Detection precision graph (training chair1, detection scene03) . . . . .	87
5.3	Detection precision map(training chair1, detection scene03) . . . . .	87
5.4	Detection precision on the object hypothesis with minimum distance to the ground truth on scene3 . . . . .	89
5.5	Detection precision on the first object hypothesis on scene3 . . . . .	90
5.6	Comparison of training cases in classification results . . . . .	91
5.7	Average recall on table objects . . . . .	93
5.8	Average weights for training case A . . . . .	95
5.9	Average weights for training case B . . . . .	96
5.10	Average distance of true positives to the ground truth for the Stan- ford dataset scenes . . . . .	97
5.11	Recall on the Stanford dataset scenes . . . . .	98
5.12	Average orientation distances over all detected scenes . . . . .	99
A.1	The developed Training-GUI . . . . .	122



# List of Notations

$\mathcal{P}$	A point cloud $\mathcal{P} = \{\mathbf{p}_i \mid \mathbf{p}_i \in \mathbb{R}^3\}$
$\mathbf{p}$	A point of a point cloud, $\mathbf{p} \in \mathcal{P}$
$\hat{\mathbf{p}}$	A modified version of $\mathbf{p}$
$\mathbf{p}_i$	The $i$ th point of a point cloud
$\mathbf{n}$	A normal vector
$\mathcal{N}_p$	The neighborhood of point $\mathbf{p}$
$\mathcal{N}_p^r$	The spherical neighborhood of point $\mathbf{p}$ in a radius $r \in \mathbb{R}$
$\mathcal{N}_p^k$	The $k$ -neighborhood of point $\mathbf{p}$ , $k \in \mathbb{N}$
$M^{m \times n}$	A matrix of size $m \times n$
$F_{tr}$	Features computed during training
${}^c F_{tr}$	Features for a specific class $c$ computed during training
${}^c f_i$	A feature computed on a training model from class $c$
$F_{det}$	Features computed during detection
$f_i$	A feature on a point cloud
$\Sigma$	A covariance matrix
$c$	A class id
$\mathbf{c}_j$	A codeword from a codebook $C$



# Chapter 1

## Introduction

For a long time, object recognition has been one of the big problems in computer vision. It has been addressed in a variety of papers and represents a well investigated topic in computer science. Many types of algorithms have been developed, each of which focuses on a different aspect and use case. Applications include optical character recognition to facilitate the automated conversion of scanned text documents into computer-readable files, as well as counting cars on traffic lanes to predict traffic jams. Object recognition has traditionally been using images captured with conventional cameras that implement the pin hole concept. These images are easy to acquire and the sensor devices are cheap. However, they suffer from a more general issue, as the projection of real world objects as seen through the camera lens onto the image sensor plane discards the depth information for each pixel.

This information, however, can greatly enhance object recognition. It allows for estimating the distance of objects from the sensor, their physical dimensions, and shape. More importantly, this information is by design independent from lighting conditions, as the distance to an object does not change when it is being lit under different conditions. In order to use additional depth information, different types of sensor devices have been developed. These include laser range sensors implementing LIDAR technology and depth sensors like Time-of-Flight cameras. In the past few years, another sensor type has become widely available which uses infrared structured light patterns projected onto the scene to compute the depth information. This technology has been implemented in the Kinect hardware developed by Microsoft for use in its Xbox 360 video game console. This consumer device has been produced in large numbers and is sold at a substantially lower price than comparable depth sensors.

With these types of sensors, the world can be reconstructed by generating a three dimensional (3D) point cloud containing the appropriate distances between object points. Algorithms operating on 3D point clouds instead of images are thus

able to use this information to detect objects in a more generic way, independently from camera position, object pose and lighting conditions.

## 1.1 Problem Statement

Finding objects can be reduced to finding instances of class models in yet unclassified input data. While the class model can be trained prior to detection by using one or multiple representations of class instances, they need to be represented in a way that allows for detecting different instances of the same class under different conditions. This process of training is called *supervised training*, since the classes to be detected and their appearance are known prior to detection and are represented by training data. The training data then needs to be analyzed and processed in an appropriate way and deconstructed into a model that enables the detection of different object instances of the same object class under different conditions. Ideally, the object to be detected is identical to the object model that has been trained. However, most often this is not the case.

The training stage can be considered a controlled environment. The objects used for training are known and can be generated in a way that best fits the needs of the following training algorithm. It is not necessary to cope with problems arising from impartial data, noise from the sensor or occluded objects. While these problems naturally occur when capturing any data with a sensor, they can be avoided, e.g. by combining a variety of measurements into one big picture or by manually modelling the required training model. The first method reduces the effects of noise by taking into account multiple measurements, thus integrating measurements over time and reducing noise. It is also possible to extend information about object parts that have not been available in previous measurements into the complete object model. The latter method, however, generates a noise-free object model by design, since a designer or artist puts their own knowledge into the model. This method is highly dependent on the working accuracy of the designer and the compliance of the artificial model with the real world model.

Object detection in general has a variety of applications. These can be subdivided into the following categories that build upon one another:

- Detecting the presence or absence of an object  
Given input data like an image captured with a camera, the goal is to find proof that the image contains one or multiple instances of the training models.
- Determining the number of occurrences of an object instance  
Given an input image, the goal is to detect the number of object instances of training models.



- Detecting the locations of object instances  
Given an input image, the goal is to detect yet unknown object instances of the training model in the input data and output the detected object locations.
- Detecting and segmenting object instances  
Given an input image, the goal is to detect unknown object instances of the training model in the input data and output both the detected object locations as well as the precise object that has been segmented from the input data. In addition, this category can also include determining the dimensions of the object as an oriented bounding box.

While only detecting the presence or absence of an object can work with just high-level information about the model and testing the conformity with extracted high-level information from the input image, determining the location of an object needs additional information. Most importantly, recent methods extract low-level information from important object parts and use spatial relations between object parts and the training object. This approach has been motivated by research in biological vision and adapted to two dimensional image processing [CDF<sup>+</sup>04] [LS03]. According to these theories, the representation used for object detection in human vision consists of the individual parts that compose an object and structural relationships between these parts in order to define the geometry [AR02].

## 1.2 Motivation

The Active Vision Group (AGAS) at the University of Koblenz-Landau constructed the autonomous service robot *LISA* (Figure 1.1). It has been developed by students and research associates in the course of several practical laboratory courses and is used to participate in the RoboCup@Home league. The @Home league of the RoboCup has been developed to investigate more sophisticated methods for robotic interaction. The initial RoboCup league aims at boosting the development of robotic algorithms in the context of a soccer game. The ambition is that by 2050, a team of robot soccer players will be able to win a standard soccer game against a team of human soccer players. While this task certainly requires sophisticated methods in various research areas, it is still conducted in a controlled environment. The superior motivation in robotic research is ignored, i.e. developing a robot interfacing with and assisting humans in any possible way. The world such a robot would need to interface with is compared to a soccer game more complex and unpredictable. The goal is to introduce the robots to a *dynamic and unstructured* environment instead of a *dynamic and structured* environment [Zan07]. The high-level purpose for a service robot is to provide practical help to elderly or disabled people in the household, as well as to simplify everyday life and

to assist humans in general. The robot thus needs a way of interfacing with the user through native language processing, understanding commands, navigating in indoor environments and grabbing and depositing objects with its built-in manipulator. The task of building a service robot involves many algorithms that need to be able to handle general purpose input data.

In this context, the robot needs to be able to detect various kinds of objects, including windows, furniture and fixtures. Given a request, such as placing a bottle of juice inside the refrigerator, the robot needs to incorporate high-level knowledge about the environment and extract the required dynamical information from sensor data in order to execute the request properly. Detected objects can be only partially visible to the sensor device, they can be arbitrarily oriented in the environment and captured data can suffer from noise. This provides a challenging task for object detection and requires open and robust methods. Inspired by recent achievements in the area of two dimensional image processing, this thesis aims at implementing an appropriate approach and adapting a robust object detection algorithm to the requirements of the current setting. The object detection task is restrained to the task of detecting furniture in indoor environments, since this is the most common and fundamental challenge in this context.

Referring to the thesis title, a *probabilistic* approach to object recognition is used, in which each object hypothesis is supported by a number of probabilities. Objects are detected at regions where maxima in the associated probability density function occur. By using a probabilistic approach, variations in the process can be compensated.

### 1.3 Goal

Motivated by recent 2D object recognition methods, this thesis aims at adapting the chosen method to the task of 3D object recognition for the service robot LISA. Given the available sensor devices, incorporating 3D data acquired by a depth sensor promises to improve precision and robustness of object detection algorithms. In practice, LISA has a *Microsoft Kinect* camera mounted on top, which makes it the first choice for the proposed algorithm. Following state-of-the-art approaches to object detection, a suitable algorithm is chosen and will be



**Figure 1.1:** The service robot LISA [SKM<sup>+</sup>13]

adapted to 3D input data. As a consequence, existing methods need to be revised and appropriate methodologies need to be matched to the special requirements. In the evaluation, this thesis will also investigate how the requirements need to be changed for the current task and how the proposed algorithm behaves in terms of precision, robustness and runtime.

## 1.4 Implementation

The developed object recognition system is intended for use in the LISA service robot. To employ modularity, the algorithms are encapsulated in a generic framework and deployed as a library. Communication with robotic platforms is achieved by incorporating the library in the Robot Operating System (ROS) [QGC<sup>+</sup>] and providing interfaces to trigger the object recognition process. The training procedure was enabled by developing an additional training application using the Qt graphical library, in which models can be loaded and the training process can be started. The system and all of its components has been developed in C++ for the GNU/Linux operating system, while the use of external open-source libraries simplifies potential porting to other operating systems later on. Standard algorithms and methods were implemented using the Point Cloud Library (PCL) [RC11] in order to assist the development process. Finally, the developed object recognition system and all of the accompanying applications are published under the terms of the GNU General Public License.

## 1.5 Outline

The outline of this thesis is as follows:

### 1 Introduction

The introduction gives an overview about the general problem and defines the goal for this thesis.

### 2 State of the Art

Existing approaches to object recognition that build the base for the developed algorithm are presented and related fundamental algorithms are introduced.

### 3 Creating Implicit 3D Representations

The training methods are described in detail, building the object model for the following recognition procedure.

#### 4 Probabilistic Object Recognition

Using the previously trained object model, a probabilistic approach to object recognition is described.

#### 5 Evaluation

The proposed algorithm is evaluated for accuracy and robustness using representative test cases.

#### 6 Conclusion

The results are summarized and implications on the use and fields of application are drawn, and further prospects are discussed.

#### A Appendix

Additional information is given considering the proposed reference implementation and notes on the practical usage.

# Chapter 2

## State of the Art

This chapter describes state-of-the-art approaches to object recognition, on which this thesis will be based. Existing methods are split into two categories. Approaches to object recognition and classification that describe effective methods are discussed in Section 2.1. These methods build the base for the proposed algorithm. Existing methods and tools that are needed by the proposed method are described in Section 2.2 and Section 2.3.

### 2.1 Implicit Shape Model

The Implicit Shape Model approach has recently become popular and describes a method to general object recognition. As the name suggests, an object is not described by a direct representation of its shape. In fact, an *implicit* representation is build, which enables the algorithm to cope with shape variations, noise and occlusion. The precise method, however, is based on earlier works in the context of text classification.

#### 2.1.1 Bag-Of-Words

In [Joa98], Joachims presents a method to categorize documents into a fixed number of classes in the context of online information retrieval. The goal is to automatically classify text documents, e.g. to find information on the web or filter news stories by personal preferences. A text document is allowed to be classified into either multiple categories, exactly one or no category at all. The classification process is modelled with a binary classification problem per category. While building such classifiers manually is difficult and time consuming, a supervised learning mechanism is proposed in order to automate the process. In a first step, the document is converted into a representation that can be used by the learning

algorithm. Prior research indicates that the occurrence of words in a document proves to be a feature from which the importance of the document can be derived, while their position in the text is of minor value. Thus, following the *bag-of-words* concept, a feature vector is constructed by mapping the distinct words inside a document to the number of their occurrence. This high dimensional feature vector is then used to train a *Support Vector Machine (SVM)* for text categorization, as SVM's are well suited to deal with large feature vectors and the feature space that the text classification subject poses [Vap99] [CV95].

Similarly, in [NMTM98], Nigam et al. propose a text classification method that aims at classifying text documents into a fixed number of categories with high precision without the need to provide a large amount of correctly labeled training documents. In contrast, they use a number of unlabeled documents to augment the labeled training documents and boost the overall precision. Using the labeled training documents to estimate the classification of unlabeled documents, co-occurrences of words can provide further information about the joint probability distribution of words within the documents. While a naive Bayes classifier [DP97] was trained using the frequency of words as features, the Expectation-Maximization (EM) [DLR77] algorithm was used to incorporate information from unlabeled training documents into the classification process. The authors proved the superiority of their combined approach and showed that their approach achieves a higher accuracy with fewer labeled training documents in contrast to using the Bayer classifier on the labeled training data alone.

## 2.1.2 Object Recognition

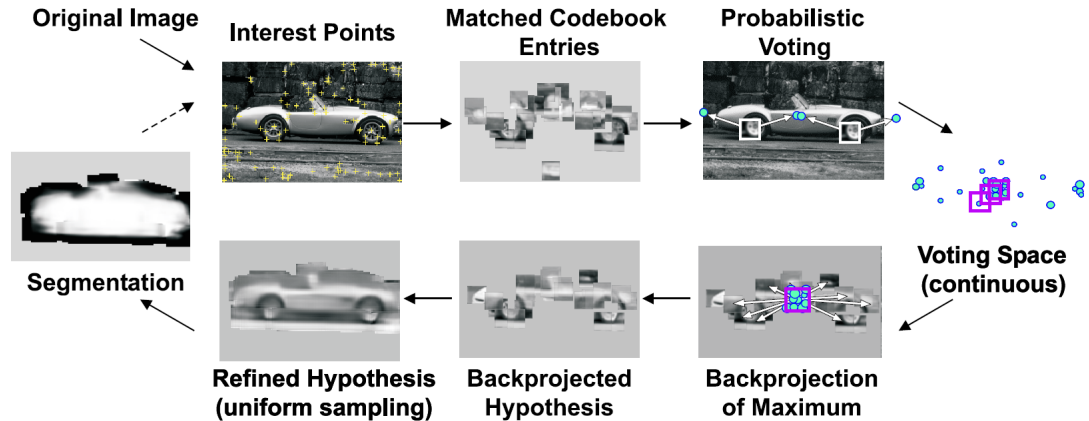
Inspired by previous results in text classification, Csurka et al. introduced the *bag-of-keypoints* approach for generic visual categorization [CDF<sup>+</sup>04]. Using the Harris affine detector [MS02], interest points are extracted from an input image. The area around the interest points is then characterized by SIFT descriptors [Low99]. The SIFT descriptor is supposed to be robust toward noise and has a high feature dimension, which makes it potentially more descriptive compared to other descriptors. Additionally, it is justified to using the Euclidean norm to compare the descriptors in feature space. In order to reduce the complexity of the search process, a visual vocabulary is proposed that is being constructed by clustering descriptors into representative *keypoints* that can be used in the detection process. Clustering is performed using a K-Means clustering algorithm and assigning each training feature a cluster center to create the keypoint within the vocabulary. In the categorization process, features from an input image are matched toward the vocabulary and assigned their closest keypoint in the feature space. Additionally, the number of occurrences of the keypoint is counted, thus creating the bag-of-keypoints. The input image is then being categorized by (1)

applying the naive Bayes classifier and choosing the category with the highest score and (2) by using an SVM for categorization and reducing the problem formulation into a binary classification problem. In the experiments, the authors showed that their approach is robust toward noise and is able to categorize well without using geometric information with the bag-of-keypoints approach. Results with the SVM classifier were superior compared to using the naive Bayes classifier.

Based on these previous works, Leibe and Schiele first introduced the initial *Implicit Shape Model* approach to recognize unknown objects in a real world scene using a probabilistic formulation [LS03] [LLS04] [LLS06] [LLS08]. Previous approaches to object recognition have primarily focused on object segmentation and the extraction of low-level features. Driven by research in human vision, the authors investigated the use of high-level features in order to group image features and drive the segmentation process.

The object is represented by a number of images. To cope with changes in the viewpoint with respect to the object, a series of training images from several views is used in the training process. Starting with the training images, keypoints on prominent locations are extracted using the Harris corner detector. Image patches of a fixed size of 25 x 25 pixels are then extracted around the keypoint positions which represent the local neighborhood. The patches are then clustered by employing a hierarchical agglomerative clustering algorithm. The clustering process starts with each image patch as a separate cluster. Two separate clusters are linked together if the distance between their corresponding image patches is below a threshold. Similarity between image patches is determined using the Normalized Greyscale Correlation (NGC) measure. By considering all image patches within the clusters, this approach guarantees that the clusters stay compact while being separated from one another. Using this procedure, the authors were able to reduce the amount of image patches by 70%. The resulting clusters contain only image patches which are visually similar, thus can be considered *codewords* on the object. A codeword is therefore a visual pattern on the object that might occur on different locations, however in similar appearance. The set of codewords is then referenced as codebook entries in a *codebook*.

The codebook generation process yields a codebook (also termed *alphabet*) of visual appearances. However, the codebook alone does not contain any spatial information about the codewords. While the codebook only approximates the local visual properties of the object, determining the position in an unknown scene requires additional information. Following the codebook generation, the extracted image patches are matched against the codebook. All image patches whose similarity to a codeword is above a threshold are activated. The authors suggest to use the same threshold in the activation process as in the agglomerative clustering process. For each activated image patch, the patch position is stored with



**Figure 2.1:** Recognition procedure from [LS03]. Given an input image, image patches are extracted around interest points and matched with previously trained codebook entries. Matching entries then cast votes for possible object positions into a continuous voting space. Maxima in the voting space are detected and the image patches contributing to the maxima are backprojected into the image space to create the object segmentation.

respect to the object’s center. Thus, each codeword is associated with several vectors originating from a number of activated image patches and reflecting where this codeword can be found on the object. The codebook and its corresponding activation vectors then form the Implicit Shape Model (ISM) for the given object class.

During object recognition, the ISM is employed in a probabilistic framework based on a Generalized Hough Transform [Bal81]. After image patches have been extracted from a test image in the same manner as during training, the resulting patches are again matched with the codebook. Activated codewords are then queried to obtain the corresponding activation vectors that have been generated during training. While the activation vectors have initially been generated from image patch locations in relation to the known object center, this information is used in the detection stage to derive hypotheses for object locations from image patch locations. A codeword then casts a number of votes for a possible object location into a voting space. Hypotheses are created by analyzing the voting space for maxima using Mean-Shift Mode Estimation [Che95]. However, instead of using a uniform vote weight in the voting space, the voting procedure utilizes a probabilistic formulation. For every vote, a weight encodes the probability that this vote is supposed to represent the correct object location. The sum of votes building the voting space then contributes to a number of object hypotheses which are found by employing Mean-Shift Mode Estimation (Figure 2.1).

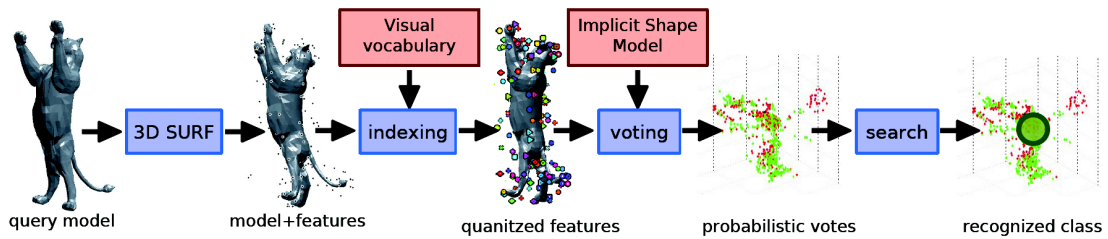


Furthermore, the authors use the object hypotheses found in the previous step to segment the object from the background. For each image patch, the activated codeword therefore also stores a list of segmentation masks for each object location it occurs in, while the mask is derived from a known figure-ground segmentation mask for the given training image. When an object hypothesis has been detected in the voting space, all votes that contributed to this hypothesis are collected and the corresponding image patches are backprojected to their original position. As part of the segmentation process, the segmentation masks stored for each activated codeword are applied to the input image and combined into a pixel-wise probability. A threshold that represents the desired minimum confidence value is then applied on the resulting confidence image in order to obtain a final object segmentation.

Knopp et al. describe an algorithm for three dimensional object classification [KPW<sup>+</sup>10]. Given a query object, their task is to find the proper object class learned from a shape database.

The authors use an extension to the SURF descriptor [BTG06] to three dimensional shapes. In an initial step, the shape is fitted inside a voxel grid of size  $256^3$  and reduced to a voxelized shape by intersecting the object faces with the voxel bins. For all voxel bins, a *saliency measure* is computed over three octaves derived from the second order derivative for the current octave. Using a non-maximum suppression technique, the saliency values are pruned to compute a number of unique features. In order to obtain a rotation invariant descriptor, a local reference frame is computed by analyzing Haar-wavelet responses around each interest point. By sampling a  $3 \times 3 \times 3$  grid relative to the local reference frame and computing a description vector of Haar-wavelet responses for each grid bin, the final descriptor is then composed from the wavelet description vectors over all grid bins. This results in a  $3 \times 3 \times 3 \times 6 = 162$  dimensional feature vector at the interest point. The feature description stores the interest point location, a scale value extracted from the saliency measure and the SURF feature vector.

Using this descriptor, the Implicit Shape Model approach is used to create a model for each class by generating a visual vocabulary from the training data. The input feature vectors on the training model are clustered using the K-Means algorithm to reduce the dimensionality of the matching process. As a heuristic, the number of clusters is determined by 10% of the number of input features. The visual words inside the vocabulary are constructed as the cluster centers from the K-Means clustering. Following the ISM approach, votes are generated for each visual word by describing the offset from the object center position to the location where the visual word has been activated. However, while assigning the feature only with the closest visual word may seem naturally, more noise-robustness can be achieved by matching it with all visual words that have a descriptor distance below



**Figure 2.2:** Recognition procedure from [KPW<sup>+</sup>10]. Given a query model, 3D SURF features are extracted and matched with the previously trained visual vocabulary. Using the activation distribution contained in the Implicit Shape Model, each visual word then casts votes into a voting space, which is analyzed to find the location with maximum density in order to detect the specific class.

a threshold. This is especially useful if the distance between visual words is small. This matching process creates a distribution for each visual word, which may cast votes for multiple classes and object positions. Each vote is also associated with the feature scale from the feature that generated the vote, and the object scale.

Given a feature on a test model at a specific location, the feature is matched against the visual vocabulary and the matching visual word casts its associated votes in relation to the feature location into a voting space. The votes therefore support the occurrence of a specific object class at a specific location. Scale invariance is achieved by taking into account a relative scale value, derived from the feature scale and the object scale. If the test model precisely matches the training object, all votes will be generated at similar positions in the voting space, thus making a strong cluster of votes. If the test model does not match the training object at all, the features will not match the visual words very well, therefore scattering around in the voting space.

Instead of casting the votes directly into the voting space, the votes are further weighted to account for feature-specific variations. Not all features may contribute equally to the detection of a specific class instance, while also statistical inequalities cause the clusters in the voting space not to be comparable. The votes are therefore weighted with two different weights. The *statistical weight* weights all votes cast by a visual word and makes the votes invariant to statistical variation, e.g. the number of training samples in the class or the number of votes associated with the visual word. The *learned weight* can already be determined during training and takes into account that a feature can cast multiple votes on different locations, while only one object position is correct. This way, features with votes closer to the actual object center are weighted higher as features that created votes far from the object center. The final vote weight is the combination of the two separate weights.

Detecting the class of a test object requires analyzing the 5D voting space (object position, scale and class) for clusters. The authors describe two mechanisms to determine the class of an unknown object. The *cube searching* approach discretizes the voting space into bins. Each vote that falls into a bin can also contribute to neighboring bins by weighting the influence by a Gaussian. Recognizing the position and class of an object is performed by analyzing the discretized voting space for maxima. This approach allows to cope with noisy and partial test objects. However, when given a clean test object, the *distance to shape center* approach simplifies class recognition by weighting each vote by a Gaussian, computed from the distance to the center. Class recognition is then performed by finding the class that best matches the given object center (Figure 2.2).

The authors do not address rotation in [KPW<sup>+</sup>10], but discuss approaches to solving rotation invariant object recognition for hough transform based methods in general [KPVG10]. The problem is divided into three categories, depending on the additional information available for the input data. In case a local reference frame is available for each feature, *point voting* transfers an object-specific vote from the global reference frame into the local reference frame computed for the corresponding feature during training. Given that reference frames can be determined robustly, the vote can be transferred back into the global reference frame at detection, by incorporating the local reference frame computed for the corresponding feature. Since local reference frames are subject to noise, *circle voting* can be performed when only normal information is available. Since a normal does not span a reference frame, the exact location for the vote is unknown. However, the vote position can be confined to a circle around the feature position with the normal direction. During voting, the circle can then be subsampled with appropriate resolution. With many features and each associated with a number of votes, the voting space will be covered with a multitude of subsampled circles. Finding maxima still corresponds to finding locations in the voting space with maximum density, since vote circles for matching object hypotheses will intersect at the estimated object centers. Even when no additional information like normals or local reference frames is available at all, *sphere voting* can still confine the vote position to a position on a sphere. Regions with high density in the voting space are then created by voting sphere intersections.

## 2.2 Features

Based on the methods described in Section 2.1, the proposed algorithm needs to compute features on 3D point cloud data. This section therefore describes required existing methods for feature extraction, which will be used later on in the following sections.

### 2.2.1 Normals

The captured point cloud, whether it contains the object model to be trained or the scene in which objects are to be detected, does not initially contain additional information besides point locations. Most following algorithms, however, rely on normal information. The surface normal at a point  $\mathbf{p}$  is given by the normal  $\mathbf{n}_p$  of the tangent plane at this point. Given the  $k$ -neighborhood of the query point  $\mathbf{p}$ , the distance from each point  $\mathbf{p}_i$  within the neighborhood to the tangent plane with the normal  $\mathbf{n}_p$  is given by  $d(\mathbf{p}_i) = (\mathbf{p}_i - \mathbf{p}) \cdot \mathbf{n}_p$ . Estimating the tangent plane can then be formulated as a least-square plane estimation problem [Rus09, p. 45] within the point neighborhood  $\mathcal{N}_p$  of  $\mathbf{p}$ . The covariance matrix  $\Sigma$  of  $\mathcal{N}_p$  is computed by:

$$\Sigma = \frac{1}{\|\mathcal{N}_p\|} \sum_{\mathbf{p}_i \in \mathcal{N}_p} (\mathbf{p}_i - \mathbf{p}) \cdot (\mathbf{p}_i - \mathbf{p})^T \quad (2.1)$$

The eigenvalues of the covariance matrix are real numbers  $\lambda_i \in \mathbb{R}$  and their corresponding eigenvectors  $\mathbf{v}_i$  form an orthogonal frame, corresponding to the principal components of  $\mathcal{N}_p$  [HDD<sup>+</sup>92]. If  $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$  with corresponding eigenvectors  $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ , the estimated surface normal at point  $\mathbf{p}$  is then given by  $\pm \mathbf{v}_3$ .

The sign of the normal, however, cannot be determined by analyzing the covariance matrix of the local point neighborhood only. This poses problems especially with point descriptors, which often depend on a signed normal in order to produce repeatable values. As one of the properties of the signed normal, a consistent normal orientation is desired along the whole object.

When computing normals on a point cloud that has been captured from one specific viewpoint, i.e. a point cloud captured with a Kinect camera, the normals can be oriented toward the viewpoint position. Given the viewpoint position  $\mathbf{p}_v$  and a surface normal  $\mathbf{n}_p$ , the normal is flipped according to the dot product between viewpoint position and normal:

$$\mathbf{n}_p = \begin{cases} -\mathbf{n}_p & \mathbf{n}_p \cdot \mathbf{p}_v > 0 \\ \mathbf{n}_p & \text{otherwise} \end{cases} \quad (2.2)$$

This provides an adequate way to consistently adjust the normal orientation among all points of an object and guarantees that the surface normals at each point in the point cloud are always oriented toward the camera.

As already mentioned, this only applies to a point cloud captured from a single viewpoint. At the time of training, a number of different point clouds is usually combined to provide a more complete model of the object. Thus, the aforementioned method does not work in this case. The point cloud on which normals

need to be computed rather represents a complete or partial object model, either manually modelled using a 3D graphics software, or by combining multiple scans from different viewpoints into an integrated model. In this case, normals cannot be aligned toward the camera viewpoint.

However, Hoppe, DeRose, Duchamp et al. [HDD<sup>+</sup>92] describe a way to consistently orient the normals on a set of unorganized points. First, an undirected graph is constructed by fully interconnecting all points in the point cloud. The edge weights represent the Euclidean distance between the connected points. Based on this data structure, an *Euclidean Minimum Spanning Tree (EMST)* is constructed, finding a path through the graph with minimum total edge cost, while at the same time reaching every single graph node. Converting the EMST back into a graph, every graph node is then extended with additional edges connecting the current node with its  $k$  nearest neighbors in the point cloud. The extended EMST is called *Riemannian Graph*, in which every edge is then re-weighted with the cost  $1 - |\mathbf{n}_i \cdot \mathbf{n}_j|$ , representing the vector product between the corresponding point normals. This cost guarantees that nearly parallel normals result in a low edge weight. Starting from a seed point, the Riemannian Graph is then explored using *depth first search* and normals are propagated through the graph. Whenever the graph search reaches an edge of which the target normal is inversely oriented to the source normal, the target normal is flipped. The resulting object normals are consistently oriented, based on the direction of the normal at the chosen seed point.

### 2.2.2 Keypoints

While the input data is only an accumulation of 3D points, it is necessary to find a way to reduce the dimensionality and keep only the most descriptive components, i.e. finding those parts of the point cloud that are well-described by their surrounding and suited to discriminate different areas from each other. In this context, a *keypoint* (also: *interest point*) is a single point that is characterized by unique point cloud features in its surrounding area which allow this point to be distinguishable to other points. Keypoints typically occur at locations where the surrounding area contains high detail. A keypoint detection algorithm should work invariantly to noise and clutter, changes in lighting, the perspective and capturing modalities. As such, keypoints detected on an object contained in one point cloud are supposed to occur at the same relative positions on the object in another point cloud. This property poses the fundamental concept behind object detection using local features, since it allows to compare keypoints coming from different sources and abstracting from the underlying capturing modalities.

Desired keypoints properties are:

- **Reproducibility**  
Given different scans from the same object, i.e. scans retrieved under different capturing conditions, keypoints are detected on the same relative positions on the object.
- **Uniqueness**  
Keypoints are unique, in the sense that a keypoint represents a unique part of the object and no two keypoints represent the same part. The regions around the keypoints need to contain enough detail in order to distinguish them.
- **Robustness**  
Keypoint positions are neither affected by translation, rotation and scaling nor by noise and changes in point density.
- **Efficiency**  
Keypoints can be computed efficiently, such that computing a multitude of keypoints can be performed in reasonable time.
- **Quantity**  
A large number of keypoints can be detected even on small objects.
- **Precision**  
Keypoint positions are computed at precise locations and keypoint locations computed on a different scan of the same object do not change significantly in their relative object location.

In this thesis, three different methods of keypoint detectors are discussed. The PCL implementation of the *Harris interest point detector* for 3D point clouds poses a modification to the original Harris detector for images. The 2D version of the detector takes into account the gradient directions in the direct neighborhood of the current pixel [HS88]. Starting with a window function, each pixel is associated with a matrix  $M^{2 \times 2}$  consisting of the image gradients in  $X$ ,  $Y$  and  $XY$  direction. A high image gradient correlates with a high change in image intensity in the given gradient direction at that point. Analyzing the eigenvalues of  $M$  leads to a score, with high values indicating a possible corner. Eigenvalue decomposition can be avoided analytically by computing the score as  $R = \det(M) - k \operatorname{tr}(M)^2$ , where  $k \approx 0.04$  is an empirically chosen constant. The 3D modification implemented in the PCL relies on the same idea, but instead uses normal directions. Changes in the point neighborhood are not determined by image gradients but rather by changes in the normals. For each point in the point cloud, a covariance matrix of

normal directions in the spherical neighborhood is created. The resulting corner score is computed similar to [HS88] and a high value indicates a high probability for a 3D corner.

*Intrinsic Shape Signatures* became popular recently as an alternative 3D descriptor and interest point detector [Zho09]. Firstly, a local reference frame is defined by computing a weighted covariance matrix of all points within the spherical neighborhood of the point. Each individual covariance is weighted by the inverse point density to create a representation invariant to point density variations. Since the obtained reference frame is computed from the eigenvalues of the covariance matrix, the precise axis directions pose an ambiguity of  $180^\circ$ , which results in four possible variants. Centered on a point in the point cloud, the characteristics of the point neighborhood are encoded in an occupational histogram, created from a partition in the polar coordinate system that has been aligned with the local reference frame. The spherical partitions are constructed recursively from an octahedron, yielding uniformly distributed cells on the surface of a sphere. Each cell defines a bin in the occupational histogram and is mapped to a bin label according to a lookup table. A final descriptor is created that maps each histogram bin to the number of points that fall inside the corresponding bin. Using an octahedron as basis for the spherical partitions results in a uniform cell distribution and avoids problems arising with noise in the point data. The ambiguity in the local reference frame is overcome by computing the descriptor for each of the four possible variants. Although the ISS initially describes a descriptor, analyzing the covariance matrix of the local point neighborhood is used in the PCL to create interest points. Since the computation of the local reference frame involved a weighted covariance matrix, its smallest eigenvalue is used as an indicator for large point variations in the neighborhood and the corresponding point is declared as an interest point.

Another method to extract points is *Uniform Sampling*, which has to be distinguished from the previous methods, however. Uniform sampling creates a subsampled representation of the point cloud. A voxel grid with a specified grid resolution is superimposed on the point cloud and each voxel bin is then approximated by the center of mass of all the points falling within the specified bin. If the voxel grid resolution is smaller than the average point cloud resolution, this approach creates a reduced version of the original point cloud with a lower overall resolution. This is in contrast to the above mentioned keypoint definition, since the points generated by uniform sampling do not necessarily correspond to unique features on the object. It is important to note that they do not comply to most of the stated keypoint requirements. They are not reproducible, since the voxel grid is fixed on the point cloud and variations of objects contained within do not lead to a change in the voxel grid positions (aside from variations inside the voxel bins). They are also not unique, since a unique part of the object can be represented by multiple

keypoints, according to the chosen voxel grid resolution. And lastly, noise or clutter have neither positive nor negative impact on the keypoint positions. When the voxel resolution is chosen appropriately, however, the set of keypoints generated through uniform sampling presents a sparse representation of the object and is thus suitable for object matching. Even though the keypoint generation is not influenced by object features, the probability will be high that a sufficient number of keypoints do indeed represent important object parts and are thus suited for object recognition.

### 2.2.3 Local Reference Frame

A *local reference frame (LRF)* is a coordinate system based on the characteristics of a local neighborhood for a point in a point cloud. The properties for LRFs are similar to that of keypoints and descriptors. Given an LRF computed for a specific scan at a relative position on the object, e.g. determined by a keypoint detector, the LRF for a different scan of the same object on the same relative position is expected to yield the same LRF, although the individual scans may have different characteristics regarding noise and sampling of the object. Similar to the concept of keypoints, LRF are described by the local neighborhood that is determined either by choosing  $k$  nearest neighbors or all neighbors within a radius  $r$  around the point position. Under the assumption that the entropy of each point's neighborhood is high enough, a repeatable and robust LRF for this specific position can be determined.

The representation for LRFs can be specified by a transformation matrix  $R$  of size  $3 \times 3$ , which describes the transformation to get from the global coordinate system to the LRF at the given point:

$$\hat{\mathbf{p}} = R \cdot \mathbf{p} \quad (2.3)$$

In [TSDS10], Tombari, Salti and Di Stefano described an LRF which aims at being unique and unambiguous. Similar to Section 2.2.1, they start by creating the covariance matrix of the local neighborhood of a query point. The covariance matrix is analyzed by means of *Singular Value Decomposition (SVD)* in order to get the eigenvalues and their corresponding eigenvectors. These provide information about the most dominant directions in the neighborhood  $\mathcal{N}_p^r$ , acquired at a radius  $r$  from the query point. However, they propose to use additional weights in order to reduce the influence of distant points and thus increase repeatability in presence of clutter. Accordingly, Equation 2.1 is modified as:

$$\Sigma = \frac{1}{\sum_{\mathbf{p}_i \in \mathcal{N}_p^r} (r - d_i)} \sum_{\mathbf{p}_i \in \mathcal{N}_p^r} (r - d_i) ((\mathbf{p}_i - \mathbf{p}) \cdot (\mathbf{p}_i - \mathbf{p})^T) \quad (2.4)$$



where  $d_i$  denotes the distance from a neighboring point  $\mathbf{p}_i$  to the query point  $\mathbf{p}$ , while  $r - d_i \leq 1$ . The eigenvectors of the covariance matrix are used to give an estimate about the directions of the LRF. Disambiguating the sign and therefore the axis orientations needs further analysis, however. Given the eigenvectors as starting points for the LRF axes, the precise orientation is based on the number of points in the direction of the axis, i.e. the axis is always oriented toward the denser part of the neighborhood with respect to the axis direction:

$$S_x^+ = \{i \mid d_i \leq r \wedge (\mathbf{p}_i - \mathbf{p}) \cdot \mathbf{x}^+ \geq 0\} \quad (2.5)$$

$$S_x^- = \{i \mid d_i \leq r \wedge (\mathbf{p}_i - \mathbf{p}) \cdot \mathbf{x}^- < 0\} \quad (2.6)$$

$$\mathbf{x} = \begin{cases} \mathbf{x}^+ & |S_x^+| < |S_x^-| \\ \mathbf{x}^- & \text{otherwise} \end{cases} \quad (2.7)$$

where  $\mathbf{x}^+$  denotes the original x axis and  $\mathbf{x}^-$  the flipped axis, that oriented in opposite direction. Sign disambiguation is done in the same manner for the  $\mathbf{z}$  axis, while  $\mathbf{y} = \mathbf{x} \times \mathbf{z}$ . This type of LRF has been introduced by the authors in the context of the SHOT descriptor and will be denoted by *SHOT-LRF*.

However, [Müt13] describes that this method can lead to sudden changes in the signs of LRF axes caused by slight changes in point positions or density. A proposed modification to the sign disambiguation technique is independent of point positions but rather makes use of normal and tangent directions as a feature for axis orientations. The axis is flipped if the majority of the tangents and normals within the neighborhood are pointed in the opposite direction of the current axis:

$$S = \{i \mid d_i \leq r\} \quad (2.8)$$

$$\hat{S}_x^+ = \{i \in S \mid \mathbf{x}^+ \cdot (\mathbf{p}_i - \mathbf{p}) \geq 0\} \quad (2.9)$$

$$\hat{S}_z^+ = \{i \in S \mid \mathbf{z}^+ \cdot \mathbf{n}_i \geq 0\} \quad (2.10)$$

$$\mathbf{x} = \begin{cases} \mathbf{x}^+ & \|\hat{S}_x^+\| \geq \frac{\|S\|}{2} \\ \mathbf{x}^- & \text{otherwise} \end{cases} \quad (2.11)$$

$$\mathbf{z} = \begin{cases} \mathbf{z}^+ & \|\hat{S}_z^+\| \geq \frac{\|S\|}{2} \\ \mathbf{z}^- & \text{otherwise} \end{cases} \quad (2.12)$$

$$\mathbf{y} = \mathbf{x} \times \mathbf{z} \quad (2.13)$$

This method works independently from the point density in the neighborhood and will be denoted by *SHOT-NA-LRF* (*SHOT Normal-Aligned LRF*).

After computing the reference frame, the axis vectors  $\mathbf{x}$ ,  $\mathbf{y}$  and  $\mathbf{z}$  pose the orthonormal basis vectors of the rotation matrix:

$$R = \begin{bmatrix} \mathbf{x}^T \\ \mathbf{y}^T \\ \mathbf{z}^T \end{bmatrix} = \begin{bmatrix} x_x & x_y & x_z \\ y_x & y_y & y_z \\ z_x & z_y & z_z \end{bmatrix} \quad (2.14)$$

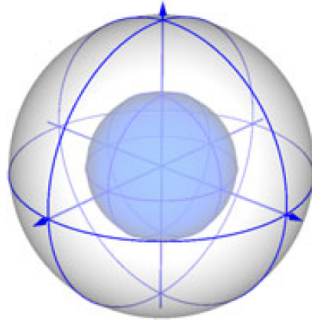
Rotating a point with matrix  $R$  as shown in Equation (2.3) computes the transformation from the global to the local reference frame. Using  $R^{-1} = R^T$  instead describes the inverse rotation.

## 2.2.4 Descriptors

Given a number of keypoint positions on the model, the high-level goal is producing a representation suitable for finding correspondances between points. While methods exist that rely on global features for object detection, the proposed method uses low-level features to create an object model for detection. Low-level features describe the local neighborhood of the point of interest and enable finding correspondences between similar points. While the local neighborhood can be represented as a subset of the original point cloud, extracted by a sphere with a radius around the interest point, using the neighboring points directly as a descriptor does not prove beneficial. The purpose of a descriptor lies in reducing the dimensionality of the search problem and condensing the most descriptive characteristics in the neighborhood to a comparable feature vector. Therefore, given any of two descriptors of unknown relative positions on an object, a successful match between those two indicates a high probability that the locations on the object correspond to each other.

Since all local descriptors compute features based on a local neighborhood around a query point, it is crucial to define a method for fast nearest neighbor computation. For 2D images, determining nearest neighbors of a pixel can be performed easily by exploiting the regular structure of the image. The direct neighbors are therefore given by all pixels enframing the query pixel. Most point clouds, however, do not contain structured information and are merely an unordered collection of 3D points, i.e. querying a points  $k$  nearest neighbors would require testing each point in the point cloud for the distance to the query point. An efficient method for fast nearest neighbor computation in point clouds can be achieved by using a *k-d tree*, which recursively partitions a point cloud into half-spaces with equal numbers of points [Ben75]. Exploiting the structure of the k-d tree then allows for fast nearest neighbor queries without the need to test each point.

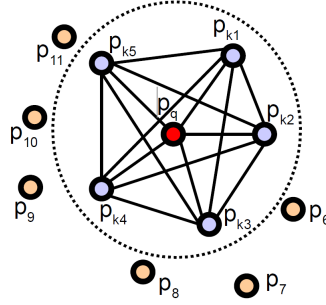
As pointed out by [TSDS10], two types of descriptors can be distinguished. Centered around a point in the point cloud, histogram-based descriptors accumulate geometrical or topological features in the surrounding of the point in a



**Figure 2.3:** Support sphere of the SHOT descriptor [TSDS10]. Note that for convenience only 4 azimuth subdivisions are shown.

histogram in a domain-specific feature space. These methods usually require the definition of a reference frame or a reference axis to put the histogram in a repeatable context. Descriptors based on histogram comparisons typically compress the geometrical structure into a more descriptive and less-dimensional feature vector that can be used for matching. In contrast to histograms, signature-based methods do not encode only one geometrical property into a histogram. A signature is rather an aggregation of individual measurements computed on the surrounding of each interest point and put in the context of a unique invariant reference frame.

The *SHOT (Signatures of Histograms of Orientations)* descriptor has been introduced in [TSDS10]. The authors began by outlining the need for a unique LRF and proposed the SHOT-LRF, as described in Section 2.2.3. Besides robustness to noise and clutter, a 3D scene can especially suffer from variations in local point densities, amongst others induced by the modalities of the underlying capturing device. When a scene is typically captured using a camera-like device with a single viewpoint, the point density will decrease with the distance from the sensor. Therefore, comparing descriptors originating from points with different distances from the sensor needs to cope with density variations. Noise and clutter, however, are disturbance factors that occur in most other cases as well. Inspired by the success of the SIFT descriptor for 2D images, SHOT adapts the use of local histograms placed within the point support. Centered around the interest point and aligned to the previously computed SHOT-LRF, the support with a given radius is spanned by a sphere in which each included point is assigned a normal direction. Instead of using the point positions directly, normals provide a good way to describe different regions, yet being rotation invariant. At first, the support sphere is subdivided into different sections along the three axes (azimuth, elevation and radial). The authors propose to use 8 azimuth divisions, 2 elevation divisions and 2 radial divisions, resulting in 32 sections on the support sphere (Figure 2.3). In each of the sections, a histogram is computed from the angle  $\theta_i$



**Figure 2.4:** Illustration of point pairs within a local neighborhood used for PFH computation [Rus09]. The PFH descriptor for the center point is computed from all point pairs within the support.

between the normal at each neighboring point located inside the section and the normal at the center point of the sphere. Each of the points can contribute to a bin in the histogram based on its normal difference to the center point. However, small differences between orthogonal vectors should result in a more descriptive representation than differences between parallel vectors. The assumption is that the most descriptive power is given when high variations in the local neighborhood lead to high difference values to the center point normal, while planar regions are less distinctive. Therefore, the section’s local histogram is computed over  $\cos\theta_i$ , since it transforms the non-uniform distribution of angles into a uniform representation suited for the accumulation in a histogram. To avoid boundary effects, interpolation is further performed between corresponding histogram bins of neighboring sections. Invariance is achieved by normalizing the whole descriptor to sum up to 1 while accounting for the local point density. Since the SHOT descriptor only relies on precomputed normals and a fast nearest neighbor query, a large number of descriptors on a dataset can be computed efficiently.

*Point Feature Histograms (PFH)* have been proposed by [RMBB08] and build a histogram over several independent features for a point in the point cloud by analyzing all possible point pair combinations within the support (Figure 2.4). For every point pair  $\mathbf{p}_i$  and  $\mathbf{p}_j$  ( $i \neq j$ ) with respective normals  $\mathbf{n}_i$  and  $\mathbf{n}_j$ , they are named source point  $\mathbf{p}_s$  and target point  $\mathbf{p}_t$  according to the angle between their respective normals  $\mathbf{n}_s$  and  $\mathbf{n}_t$ , such that the point with the smallest angle is chosen as the source point. For each point pair, a reference frame is defined by  $\mathbf{u} = \mathbf{n}_s$ ,  $\mathbf{v} = (\mathbf{p}_t - \mathbf{p}_s) \times \mathbf{u}$  and  $\mathbf{w} = \mathbf{u} \times \mathbf{v}$ .

Given all possible point pairs within the neighborhood, four independent features are defined by:

$$f_1 = \mathbf{v} \cdot \mathbf{n}_t \quad (2.15)$$

$$f_2 = \|\mathbf{p}_t - \mathbf{p}_s\| \quad (2.16)$$

$$f_3 = \mathbf{u} \cdot \frac{\mathbf{p}_t - \mathbf{p}_s}{f_2} \quad (2.17)$$

$$f_4 = \text{atan}(\mathbf{w} \cdot \mathbf{n}_t, \mathbf{u} \cdot \mathbf{n}_t) \quad (2.18)$$

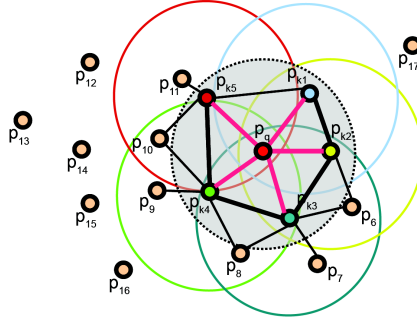
The features  $f_1$  and  $f_3$  encode the angles between the pair’s reference system and the characterizing vectors of the pair,  $f_2$  represents the distance between the points and  $f_4$  encodes the angle between the projection of the target normal  $\mathbf{n}_t$  onto the plane formed by  $\mathbf{u}$  and  $\mathbf{w}$ , and the vector  $\mathbf{w}$ . The domain of these features is well-characterized. By splitting each of the feature’s domains into half according to the center of their definition interval, each point pair then contributes to a bin in the histogram given by the bin index  $idx$ :

$$idx = \sum_{i=1}^4 2^{i-1} \text{step}(s_i, f_i) \quad (2.19)$$

$$\text{step}(s_i, f_i) = \begin{cases} 0 & f_i < s_i \\ 1 & \text{otherwise} \end{cases} \quad (2.20)$$

The resulting histogram has  $2^4$  bins and encodes the mean surface curvature of a point. More specifically, it represents the percentage of pairs that share the same feature category determined by the function  $\text{step}(s_i, f_i)$ , where  $s_i$  corresponds to the center of the feature domain. However, the authors showed that discarding  $f_2$  in the feature computation increases the descriptive power when computed on 2.5D data, i.e. data captured with a camera-like depth device, where the point density decreases with the distance to the sensor [RBB09].

*Fast Point Feature Histograms (FPFH)* are a modification to the PFH descriptor as proposed in [RBB09] and are faster to compute, however at the price of a slightly reduced descriptive power. At first, the triplet of features  $\langle f_1, f_3, f_4 \rangle$  is computed for the query point and its nearest neighbors only, instead of all point pairs within the support. The resulting histogram is then called *Simplified Point Feature Histogram (SPFH)*. Secondly, the nearest neighbors are re-computed for each point in the support and a new temporary support is defined for each nearest neighbor (Figure 2.5).



**Figure 2.5:** Illustration of point pair influences used for FPFH computation [Rus09]. The SPFH is first computed for the center point and its direct neighbors only. In a following step, the SPFHs are re-computed for the neighbors and their respective neighborhood to create the final descriptor.

The SPFHs for the support points are then weighted and added to form the final FPFH descriptor:

$$FPFH(\mathbf{p}_i) = SPFH(\mathbf{p}_i) + \frac{1}{k} \sum_{i=1}^k \frac{1}{\omega_k} SPFH(\mathbf{p}_k) \quad (2.21)$$

where the weight  $\omega_k$  is given by  $\|\mathbf{p}_i - \mathbf{p}_k\|$ . With the described modifications, the FPFH descriptor reduces the complexity of the PFH computation from  $O(nk^2)$  to  $O(nk)$ . Both descriptors are not comparable, though. While PFH creates a fully interconnected mesh in the support, FPFH only considers a subset, therefore missing point pairs which might contribute to the description of the neighborhood. Additionally, FPFH also computes point pairs with points from outside the original support into the histogram, although their weights reduce the influence to the final descriptor.

## 2.3 Mean-Shift Mode Estimation

The presented approach to object recognition uses a probabilistic formulation and models a probability density function by a number of independent probability measurements. Objects are recognized as regions with maximum probability density. In order to extract these maxima, [FH75] presents a method to estimate the modes of a probability density function by a recursive gradient descent algorithm. By observing the values of sample measurements around a point within a radius, the probability density function at this point can be estimated. Finding modes then involves a clustering algorithm and estimates the gradient of the probability density function.

Mean-Shift Mode Estimation is a non-parametric kernel density estimation method, meaning that the method does not imply any knowledge about the parameters of the underlying probability density function [CM02]. The method also does not imply any knowledge of the dimensionality  $d$ . Centered at a point  $\mathbf{x} \in \mathbb{R}^d$ , the method estimates the probability density by applying a kernel with a specified radius to all neighboring data points. The radius is also called kernel *bandwidth* and specifies the influence of the kernel.

Given  $n$  observations  $\mathbf{x}_i \in \mathbb{R}^d$ , the kernel density estimate for a point  $\mathbf{x} \in \mathbb{R}^d$  is given by:

$$\hat{f}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n K_H(\mathbf{x} - \mathbf{x}_i) \quad (2.22)$$

with the kernel function:

$$K_H(\mathbf{x}) = \frac{1}{\sqrt{|H|}} K \left( \frac{1}{\sqrt{|H|}} \mathbf{x} \right) \quad (2.23)$$

The matrix  $H \in \mathbb{R}^{d \times d}$  specifies the bandwidth matrix for the kernel function  $K(\mathbf{x})$ . In order to reduce complexity,  $H$  is reduced from a fully parameterized matrix to either a diagonal matrix  $H = \text{diag}(h_1^2, h_2^2, \dots, h_d^2)$  or proportional to the identity matrix as  $H = h^2 I$ . While the first method uses individual bandwidth parameters for each dimension, the latter method assumes the same bandwidth for all three dimensions which is reasonable when using the Euclidean metric and further reduces complexity. As a result of using the Euclidean norm, only one bandwidth parameter  $h > 0$  needs to be provided. Equation (2.22) thus reduces to:

$$\hat{f}(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^n K \left( \frac{\mathbf{x} - \mathbf{x}_i}{h} \right) \quad (2.24)$$

Since Equation (2.24) holds for every kernel function, a function  $K : \mathbb{R}^d \rightarrow \mathbb{R}$  is called *kernel*, if there is a function  $k : [0, \infty] \rightarrow \mathbb{R}$  such that:

$$K(\mathbf{x}) = c_k k(\|\mathbf{x}\|^2) \quad (2.25)$$

Function  $k(x)$  is then called *profile* of  $K$ . The constant  $c_k > 0$  guarantees that  $K$  integrates to 1. With this type of kernel definition,  $K$  is a radial symmetric kernel.

Inserting Equation (2.25) into Equation (2.24), the kernel density estimator can be rewritten as:

$$\hat{f}_{h,K} = \frac{c_k}{nh^d} \sum_{i=1}^n k\left(\left\|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right\|^2\right) \quad (2.26)$$

There are a number of different kernel types. However, two of the most common ones are presented here: the unit and the Gaussian (normal) kernel.

**Table 2.1:** Common profile and kernel types

The unit profile  $k_U(x)$  with corresponding unit kernel  $K_U(\mathbf{x})$ :

$$k_U(x) = \begin{cases} 1 & 0 \leq x \leq 1, x \in \mathbb{R} \\ 0 & \textit{otherwise} \end{cases}$$

$$K_U(\mathbf{x}) = \begin{cases} 1 & \|\mathbf{x}\| \leq 1, \mathbf{x} \in \mathbb{R}^d \\ 0 & \textit{otherwise} \end{cases}$$

The Gaussian profile  $k_N(x)$  with corresponding Gaussian kernel  $K_N(\mathbf{x})$ :

$$k_N(x) = \exp\left(-\frac{1}{2}x\right), x \geq 0, x \in \mathbb{R}$$

$$K_N(\mathbf{x}) = (2\pi)^{-\frac{3}{2}} \exp\left(-\frac{1}{2}\|\mathbf{x}\|^2\right)$$

The kernel specifies the influence of neighboring data points within the kernel bandwidth. The term  $\left\|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right\|^2$  of Equation (2.26) guarantees that the kernel parameter is always in the range  $[0, 1]$ , since the kernel is normalized and only defined in this range.

With this definition, it is now possible to estimate the probability density function at a specified point  $\mathbf{x}$ , measured at discrete data points  $\mathbf{x}_i$ . In order to find the maximum density regions, however, the gradient of the probability density function needs to be estimated. In contrast to most gradient descent methods, the step size is computed adaptively and does not need to be set in advance. The step size thus does not have any influence on the convergence of the algorithm, and it is proven that Mean-Shift Mode Estimation does always converge [Che95]. Assuming that the kernel profile has a derivative for every  $x \in [0, \infty)$ , the derived kernel is defined by:

$$g(x) = -k'(x) \quad (2.27)$$

$$G(\mathbf{x}) = c_g g(\|\mathbf{x}\|^2) \quad (2.28)$$



In this context,  $K(\mathbf{x})$  is called *shadow* of  $G(\mathbf{x})$  [CM02]. Deriving the kernel profile then results in the final density gradient estimator:

$$\hat{\nabla} f_{h,K}(\mathbf{x}) = \frac{2c_k}{nh^{d+2}} \left[ \sum_{i=1}^n g \left( \left\| \frac{\mathbf{x} - \mathbf{x}_i}{h} \right\|^2 \right) \right] \left[ \frac{\sum_{i=1}^n \mathbf{x}_i g \left( \left\| \frac{\mathbf{x} - \mathbf{x}_i}{h} \right\|^2 \right)}{\sum_{i=1}^n g \left( \left\| \frac{\mathbf{x} - \mathbf{x}_i}{h} \right\|^2 \right)} - \mathbf{x} \right] \quad (2.29)$$

While the first term estimates the density using kernel  $G$ , the second term is called *mean shift* and describes the difference between the current kernel position  $\mathbf{x}$  and the mean of the data points, weighted with kernel  $G$ . This is called the *mean shift vector*:

$$\mathbf{m}_{h,G}(\mathbf{x}) = \frac{\sum_{i=1}^n \mathbf{x}_i g \left( \left\| \frac{\mathbf{x} - \mathbf{x}_i}{h} \right\|^2 \right)}{\sum_{i=1}^n g \left( \left\| \frac{\mathbf{x} - \mathbf{x}_i}{h} \right\|^2 \right)} - \mathbf{x} \quad (2.30)$$

According to [Che95], the mean shift vector is always proportional to the normalized density gradient estimate, computed over kernel  $K$ , and points in the direction of the maximum density region for the chosen bandwidth. Beginning from a given starting point, successive computation of the mean shift vector yields a sequence of kernel locations  $\{\mathbf{x}^{(t)}\}_{t=1,2,\dots}$  called *trajectory*. The shape of the trajectory is controlled by the chosen kernel function, whereupon the Gaussian kernel usually creates a smoother trajectory than the unit kernel, but might need more iterations to converge:

$$\mathbf{x}^{(t)} = \frac{\sum_{i=1}^n \mathbf{x}_i g \left( \left\| \frac{\mathbf{x}^{(t-1)} - \mathbf{x}_i}{h} \right\|^2 \right)}{\sum_{i=1}^n g \left( \left\| \frac{\mathbf{x}^{(t-1)} - \mathbf{x}_i}{h} \right\|^2 \right)} \quad (2.31)$$

Mode estimation on the probability density function thus needs to find a point  $\mathbf{x}^{(t)}$  at iteration step  $(t)$  where  $\nabla \hat{f}_{h,K}(\mathbf{x}^{(t)}) = 0$ . In this case,  $\mathbf{x}^{(t)}$  is a *stationary point* in the probability density function and considered a mode of the underlying probability density function. In practice, a threshold  $\approx 0$  specifies when the algorithm is allowed to terminate.



# Chapter 3

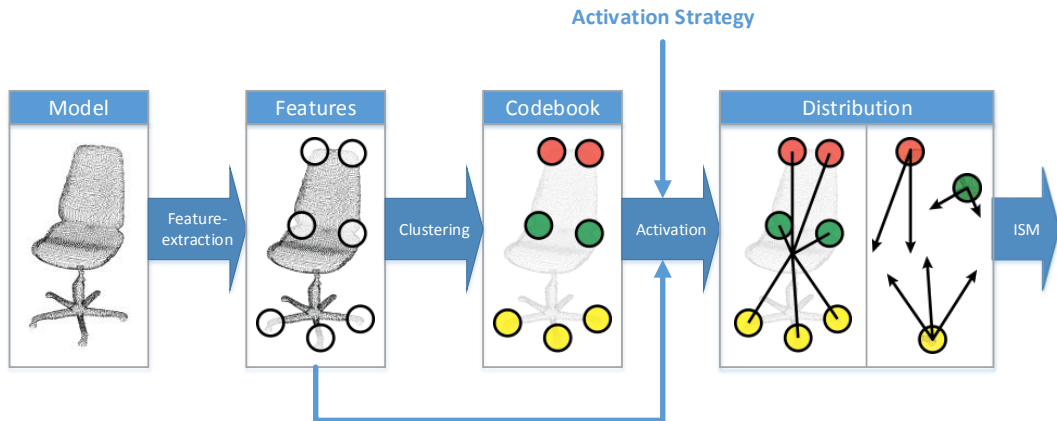
## Creating Implicit 3D Representations

Based on Chapter 2, the presented approach implements object detection based on low-level features. Similar to human vision, an implicit description for an object class is learned, comprising a number of object-specific features and their spatial relationships. Once trained, the model is then used in a detection stage on yet unclassified input data to detect hypotheses for object instances in the data. The detection is performed using a probabilistic formulation to model the recognition process.

### 3.1 Overview

The ISM framework consists of two parts.

In the training stage, the Implicit Shape Model is created in order to detect instances of multiple object classes in an unknown point cloud. In this stage, the framework is provided with pre-labeled object instances for training. After providing all available training models, interest points are being detected on the models. These correspond to locations on the training model which are supposed to represent salient features. This is based on the assumption that such interest points are sufficiently distinctive so that they will appear on unknown object instances on the same relative positions. These locations are prototypical for the specific object class and can thus be used for object detection. However, in order to compare keypoints to one another, descriptors need to be computed at the keypoint locations. A descriptor characterizes the neighborhood of the keypoint location, such that similar neighborhoods produce a similar descriptor, yet most discriminative to other locations. This allows for keypoint comparison. The keypoints are then grouped together according to their descriptors into a geometric vocab-



**Figure 3.1:** Training pipeline: Features are extracted on the initial training model and clustered by their similarity to create codewords in a codebook. The previously detected features are then matched with the codebook according to an activation strategy to create a spatial distribution of codeword locations. Codebook and activation distribution represent the trained Implicit Shape Model.

ulary, called *codebook*. The codebook contains a list of *codewords*, each of which represents a geometric word that is prototypical for the object class. However, instead of using the codebook directly to detect object instances, an *activation strategy* is employed on the keypoints and their descriptors, matching them with the codebook. A keypoint whose descriptor matches a codeword also adds the direction from the keypoint position to the object center to the ISM. Thus, the generated Implicit Shape Model consists of a list of codewords, shared between all training objects, together with a list of activation vectors for each codeword, which represent the locations where each codeword can be found on the object. The resulting ISM can then be used in the detection stage (Figure 3.1).

After training has been completed, the detection stage receives input from the sensor, attempting to detect yet unknown instances of the previously trained models in the point cloud data. At first, keypoints and descriptors are being detected on the input point cloud identically to the training stage. The keypoints are then matched against the previously trained codebook according to the activation strategy. During training, each codeword has been assigned a list of activation vectors, each of which votes for a possible object center in relation to the keypoint location. Each activated codeword now casts all of its votes into a three dimensional voting space. Each entry in the voting space corresponds to a possible object location. A probabilistic approach then assigns a weight for each vote and extracts

the most likely object locations from the voting space after the voting procedure has completed.

## 3.2 Preprocessing

Given an input sensor, sensor data needs to be acquired first. The different types of sensor devices enabling three dimensional perception include:

- Stereo cameras  
Stereo camera systems have been inspired by biological vision. Humans and most animals are able to perceive their world in 3D and estimate distances by combining two vision organs in the head, which perceive the same environment but are shifted by an offset. In a stereo camera environment, two identical cameras are pointed on the same object which are also shifted by a fixed well-known offset. The same object will then be perceived under different angles in the two images. In most cases, stereo cameras are also aligned in the same way and only shifted along a horizontal line, which reduces the mathematical part of the problem. The farther the object is away from the cameras, the smaller the angle difference between the two images will get. Thus, comparing this information allows for reconstructing the distance to the object. A software component then analyzes the images and produces a *depth map*, in which each pixel represents a distance value.
- Time-Of-Flight cameras  
A Time-Of-Flight (ToF) camera uses the speed of light to measure the distance to an object. With the speed of light as  $c = 299796 \frac{m}{s}$ , the travelling time of a light ray from the sensor to the object and back to the sensor can be determined in order to infer the distance. A ToF camera is therefore equipped with an infrared light source, emitting a light pulse at a specific time, while an aside sensor component detects the time when the reflected light pulse is measured. However, the time intervals of interest in most applications require precisely measuring the time in a *ns* range, thus additional processing is required to achieve high accuracy. After processing, the output is a dense depth map with per-pixel distance values. Additionally, some devices are able to generate a confidence score per pixel, as well as a grayscale image of the scene.
- LIDAR range finders  
The LIDAR (LIght raDAR) technology is a distance sensing technology similar to radar. It uses laser light to detect the distance to an object and measures the travelling time for a laser pulse from the time of its emission

to the detection of the reflected signal. The principle is similar to Time-Of-Flight cameras. However, using a laser allows a higher maximum detection range, while on the other hand only a single laser ray can be measured at a time. To capture multiple points, a combination of rotating mirrors is often used to successively scan the environment.

- Structured-Light sensors

Distance sensing approaches using structured light work on a combination of a projector and a camera that operate simultaneously. The setup is similar to a stereo camera system, though one camera is being replaced by a projector. While the projector casts a well-known organized pattern onto the scene, the camera records the pattern from a different angle. From the camera point of view, the perceived pattern gets distorted by the geometry of the scene. By analyzing the camera image and creating a correspondence between the original pattern and the perceived pattern, the scene geometry can be reconstructed by taking into account the offset between the camera and the projector. The result is a dense depth map, in which each pixel indicates the distance to the object beneath the pixel.

Since it is not possible to measure the dimensions of the scene directly, a distance sensing device captures a sampled version of the scene which can be transformed into a 3D point cloud  $\mathcal{P}$ :

$$\mathcal{P} = \{\mathbf{p}_i \mid \mathbf{p}_i \in \mathbb{R}^3\} \quad (3.1)$$

A point cloud is an unstructured accumulation of three dimensional data points and provides the main data source for 3D sensing applications. Depending on the type of sensor, additional information like color and normals can be stored within a point cloud.

### 3.2.1 Input Acquisition

Besides other components, the LISA robotic platform contains a *Microsoft Kinect* camera as 3D input device (Figure 3.2). The Kinect camera has been developed by Microsoft as input device for the Xbox 360 video game console and was presented to the public in November 2010. In the gaming environment, the device is placed near the TV screen and captures the scene before the sensor. Besides using a color camera for image acquisition, a depth sensing component reconstructs a dense depth map of the scene. Analysis of the depth map then detects the interacting users and estimates a representation of their body skeleton. The skeleton comprises the most important body joints and is then used to let the users interact with the console and supported games.

The depth sensing components comprise a mono CMOS imaging sensor and a static laser projector. The laser projector projects a pattern of seemingly randomly positioned and sized points onto the scene. However, the pattern is created by two special refraction panes positioned in front of the light source that deflect the laser light and thus produce the pattern. The laser is further stabilized to produce



**Figure 3.2:** The Kinect camera

equal power output, therefore the generated pattern is stable and its structure is well known. Due to the precise type of the pattern structure, the local environment of each of the points is unique and differentiable between other parts of the pattern. In order to reduce the effects of ambient light and to make the depth sensing procedure undisturbing to the user, the projector as well as the camera operate at the same IR wavelength, which is invisible to the human eye. Similar to comparable structured light systems, the pattern projected onto the scene is perceived by the imaging sensor and gets distorted by the geometry of the scene. Comparing the perceived pattern with the known pattern and taking into account the intrinsic parameters of the system's design, the geometry of the scene can be reconstructed and a point cloud can be derived from the generated depth map.

While object instances need to be detected using only one point cloud coming from a sensor, the training process needs to handle data acquisition differently. In order to generalize the object, it is preferable to have the object geometry available in good detail. Because the object can later be positioned arbitrarily inside the scene, the model used for training needs to at least incorporate details from all viewpoints from which the object could potentially be seen during detection.

- Registration

A well-known algorithm to combine multiple scans into a world model is widely used in robotics. While the robot moves through the world, 3D scanning devices like LIDAR are used to capture momentary images of the surrounding. Consecutive scans are combined by applying an *Iterative Closest Points (ICP)* algorithm, which minimizes the squared distances between the scans, given an initial pose estimate computed from odometry data.

- Kinect Fusion

The Kinect Fusion algorithm performs a real-time ICP algorithm to incorporate different Kinect scans into a more complete representation of the world. It has been first introduced in [NIH<sup>+</sup>11] and has been implemented onto

different platforms. The algorithm is optimized to run on the graphics card and therefore allows to move around with the camera and see the results in real time. Data that has been missing in one scan but is available in another scan is used to complete the real world model.

- **Manual Alignment**

Since automatic registration needs initial pose estimates to align point clouds, registration can also be performed by manually aligning individual point clouds and creating a global object representation.

### 3.3 Features

The original ISM approach for object detection in 2D [LS03] proposed to represent the local neighborhood by image patches. Assuming a fixed camera position and no rotations, a window of fixed size can be superimposed on the detected keypoint position to represent the surrounding area. Since rotation of the object or the camera cannot properly be addressed using 2D data only, abstracting from rotation invariancy seems reasonable and using image patches to describe local properties of the object is valid.

Several issues arise when adopting this approach to 3D data. In analogy to image patches with a fixed window size in 2D, a subset of the input point cloud obtained within a specified radius around the interest point represents the local neighborhood in 3D. A method to comparing and matching point cloud subsets can be defined, e.g. by computing the absolute difference between the triangulated mesh of both point clouds fixed at the same absolute position. When working with 3D, not considering rotation is not acceptable, however. The input device is allowed to move freely within the real world, just as the object can be positioned and oriented arbitrarily. Restricting the degrees-of-freedom in any case would highly constrain the operating conditions of the proposed method. The alternative lies in the concept of feature vectors that describe the local neighborhood in a suitable way and compress the characteristics to the most distinguishable components. While the amount of data in a point cloud subset is proportional to the points contained within, the dimensionality of a feature vector is independent from the structure of the neighborhood. As a side effect, the dimensionality of a feature vector is usually lower than that of a point cloud subset, thus reducing storage space and increasing performance. Additionally, descriptors often incorporate means to provide rotation invariancy themselves and thus allow for directly comparing feature vectors to compute correspondences.



Starting with each individual training model, features are computed on the training model in the following way:

- Normals  
If no normal information is available for the model, consistently oriented normals are computed.
- Keypoints  
Keypoints are detected on the model according to the chosen keypoint detector.
- Local Reference Frames  
Local reference frames are determined for each of the keypoint positions based on the local point neighborhood.
- Descriptors  
Descriptors representing properties from the local point neighborhood are extracted at the keypoint positions.

It is important to note that the computation of normals on the point cloud is critical, since most of the following feature-extracting algorithms rely on correct normal estimation. As pointed out already in Section 2.2.1, while the normal extraction is basically robust, the precise normal orientation cannot be determined using the local neighborhood alone. Depending on the chosen method to generate training models, normals can be extracted already. Since approaches using point cloud registration incorporate different scans, normals can be computed on each of the scans and oriented toward the current viewpoint. The registration step can then merge the individual normals to create consistently oriented normals on the training model. When using Kinect Fusion, correct normals are automatically extracted along with the global point cloud.

When no normal information is available prior to the training process, normals are computed using the methods described in Section 2.2.1. At first, the covariance matrix of the point neighborhood is used to estimate ambiguous normal directions. Starting from a seed point, the normal direction is then propagated on a modified minimum spanning tree, as described in [HDD<sup>+</sup>92]. The seed point is chosen as the point with maximum distance from the model center and the corresponding normal is oriented to point away from the object center. Starting with this configuration, the normal direction is propagated on the model.

After all the individual features have been computed, the condensed feature representation  $f_i$  is the local representation of an object part and is herein defined as a triple, composed of a keypoint position  ${}^{f_i}\mathbf{p}$ , a local descriptor  ${}^{f_i}\mathbf{l}$  with dimensionality  $n$  that describes the properties of the local neighborhood in relation to

the keypoint position, and a representation of the local reference frame, given by a rotation matrix  ${}^{f_i}R$ :

$$\begin{aligned} f_i &= \langle {}^{f_i}\mathbf{p}, {}^{f_i}R, {}^{f_i}\mathbf{l} \rangle & (3.2) \\ {}^{f_i}\mathbf{p} &= ({}^{f_i}p_x, {}^{f_i}p_y, {}^{f_i}p_z)^T & \in \mathbb{R}^3 \\ {}^{f_i}R &= [{}^{f_i}\mathbf{x}^T \quad {}^{f_i}\mathbf{y}^T \quad {}^{f_i}\mathbf{z}^T]^T & \in \mathbb{R}^{3 \times 3} \\ {}^{f_i}\mathbf{l} &= ({}^{f_i}l_1, {}^{f_i}l_2, \dots, {}^{f_i}l_n)^T & \in \mathbb{R}^n \end{aligned}$$

For a given class  $c$ , a feature detected on a model belonging to  $c$  is denoted by:

$${}^c f_i = \langle {}^{f_i}\mathbf{p}, {}^{f_i}R, {}^{f_i}\mathbf{l} \rangle \quad (3.3)$$

The set  $F_{tr}$  finally represents all detected features on the training models:

$$F_{tr} = \{{}^c f_i\} \quad \forall c \quad (3.4)$$

### 3.4 Clustering and Codebook Generation

After computing features on the training data, a *codebook* is created by clustering features according to their corresponding descriptors into *codewords*. The clustering is performed in the feature’s descriptor space, therefore clustering features together based on the geometrical similarity of their corresponding object regions. The resulting cluster center is then also given in the descriptor space and represents a meta descriptor, which is prototypical for a specific object characteristic. Under the assumption that the whole of an object is composed of a quantity of recurring local features reoccurring at different positions, the codebook thus constitutes all possible object characteristics, independently from their relative positions on the object. While the 2D Implicit Shape Model used image patches to describe the local properties of an object, descriptors cannot directly be considered visual representations of the underlying geometry. Codewords are therefore referred to as geometrical representations, rather than visual representations.

According to [STDS10], two types of codebooks can be distinguished. A *local codebook* (also: *separated codebook*) treats each object class individually. After computing features on each of the training models for each object class, features of each class are clustered to create a class-specific codebook. During detection, a codebook will be used for each of the object classes. It is likely that a codebook contains codewords that are similar to codewords of a different codebook for a different class. In contrast, a *global codebook* is computed over all detected features in all classes. Features from all classes and training models can contribute

to the implicit representation for a specific object class. Using a global codebook approach allows for a wider generalization. While the local codebook generalizes at inter-class level, the global codebook also generalizes at intra-class level. The codebook representation built over all object classes therefore creates a codebook containing all codewords that could possibly occur, given all training models. During detection, the codebook is then shared among all object classes. Motivated by the results provided by [STDS10], the approach presented here uses a global codebook.

The process of codebook creation has been evaluated by using different clustering methods. The *K-Means* clustering method is a form of *partitioning clustering*. These methods typically start with an initial cluster configuration given a cluster count  $k$  and iteratively relocating cluster elements between the clusters until the final configuration is reached. K-Means clustering optimizes the cluster assignments such that the  $k$ -clusters represent the means of their assigned cluster elements. Given  $\mathbf{x}_i \in X$  a set of  $n$   $d$ -dimensional data points, the K-Means algorithm divides the data set into  $k$  partitions  $S_j \in S$  by minimizing the error function:

$$E = \sum_{j=1}^k \sum_{\mathbf{x}_i \in S_j} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2 \quad (3.5)$$

The term  $\|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2$  specifies the squared distance from a cluster element  $\mathbf{x}_i$  to the cluster mean  $\boldsymbol{\mu}_j$ . This is equivalent to minimizing the *within-cluster sum of squares*. Lloyd used a K-Means algorithm for quantization in pulse-code modulation (PCM) [Llo82] and his algorithm is commonly used nowadays. In the initialization,  $k < n$  random points are chosen as cluster centers  $\boldsymbol{\mu}_j$  from the data points. In the assignment step, each data point gets assigned to its nearest center. The centers are then updated to reflect the centroid of all data points assigned to this cluster center, thus computing the cluster mean. This step changes the cluster centers from their initial random position to a new position located inside the cluster points. The algorithm iteratively continues by again assigning each data point to its nearest cluster center and recomputing the cluster mean. The algorithm terminates once the assignments do no longer change (Algorithm 1).

It is important to note that this algorithm only computes an approximated partition of the original data set. When executed twice with the same parameters, the algorithm does not necessarily return the same precise partitions, since the initial cluster centers are chosen randomly. Reducing those artifacts can be accomplished by applying the algorithm multiple times and evaluating the error function from Equation (3.5) after each cycle. The best partitioning is then given by the minimum value of the within-cluster sum of squares. Additionally, since the K-Means algorithm computes the mean of all cluster elements, it is only applicable

---

**Algorithm 1:** K-Means clustering algorithm

---

**Input:** Data points  $\mathbf{x}_i \in X$ , Cluster count  $k$ **Output:** Set of partitions  $S_j \in S$ **begin**     $M \leftarrow \{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_k \mid \boldsymbol{\mu}_i \text{ randomly chosen points from } X\}$      $S \leftarrow \{S_1, S_2, \dots, S_k \mid S_j = \emptyset\}$     **repeat**         $S_j \leftarrow \{\mathbf{x}_i \in X \mid \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2 \leq \|\mathbf{x}_i - \boldsymbol{\mu}_m\|^2 \quad \forall j \in \mathbb{N}, 1 \leq j \leq k\}$          $\boldsymbol{\mu}_j \leftarrow \frac{1}{\|S_j\|} \sum_{\mathbf{x}_i \in S_j} \mathbf{x}_i$     **until**  $S_j = S_{j-1}$ ;

---

with the Euclidean distance measure. However, in the given application, using the Euclidean distance measure is reasonable, since the chosen descriptors depend on their local reference frames and are thus elementwise comparable.

One of the main drawbacks of K-Means clustering is the choice of  $k$ , which is not trivial. Several approaches exist to determine  $k$  automatically. Simple solutions estimate  $k$  from the size of the data set, i.e.  $\|X\|$ . Several rules of thumb exist, such as  $k = \sqrt{\frac{\|X\|}{2}}$  [MKB79], or  $k = c \|X\|$ , assuming a certain percentage of the data size for  $k$  and referring to  $c$  as the cluster factor. More sophisticated heuristics apply K-Means multiple times with different values for  $k$  and analyze Equation (3.5) as a function of  $k$  [AS09]. Hartigan's index [Har75] is given by:

$$H(k) = \gamma(k) \frac{E_k - E_{k+1}}{E_{k+1}}, \quad \gamma(k) = N - k - 1 \quad (3.6)$$

The error function  $E$  is evaluated for each  $k^{(t)} = k^{(t-1)} + 1$  and the final cluster count is given by the smallest  $k$  that satisfies  $H(k) \leq \eta$ , while typically  $\eta = 10$ . More heuristics exist, as well as an adapted K-Means algorithm that does not require knowledge of the cluster count [PM00]. However, the precise choice of  $k$  is not critical, since the exact number of partitions cannot be precisely determined for the high dimensional descriptor space. Slight variations in the clusters are allowed and are not crucial to the algorithm, as long as the cluster assignment works as expected and the within-cluster sum of squares is effectively minimized. Using the above described heuristics is therefore sufficient for the current approach.

In contrast to partitioning clustering methods, which typically require knowing the number of clusters in advance, *hierarchical clustering* methods are a form of top-down agglomerative clustering [ER80]. Clustering starts with each of the data points as their own clusters. According to a distance strategy, the two most similar clusters are chosen and their elements are merged to create a bigger cluster. This process can either be continued until the cluster distance for two selected clusters

exceeds a threshold, or until only one cluster is left. In the latter case, the clustering process is modeled in a cluster graph and the cluster structure can be determined anytime by cutting the graph according to the desired threshold or cluster count. Since there is no need to later adapt the threshold on the same data, the clustering process is terminated once the threshold is reached, as depicted in Algorithm 2.

Determining which two clusters need to be merged is referred to in literature as *linkage*. While *single linkage* and *complete linkage* only consider distances between selected single elements within the clusters, *average linkage* computes the Euclidean distance between every single element between both clusters:

$$dist(S_i, S_j) = \frac{1}{\|S_i\| \|S_j\|} \sum_{n=1}^{\|S_i\|} \sum_{m=1}^{\|S_j\|} \|\mathbf{x}_{i,n} - \mathbf{x}_{j,m}\|^2 \quad (3.7)$$

---

**Algorithm 2:** Agglomerative clustering algorithm
 

---

**Input:** Data points  $\mathbf{x}_i \in X$ , Threshold  $t$

**Output:** Set of partitions  $S_j \in S$

**begin**

$S \leftarrow \{\{\mathbf{x}_1\}, \{\mathbf{x}_2\}, \dots, \{\mathbf{x}_n\}\}$

$S_{i,min} \leftarrow \emptyset$

$S_{j,min} \leftarrow \emptyset$

$d_{min} \leftarrow \infty$

**repeat**

**foreach**  $S_i \in S$  **do**

**foreach**  $S_j \in S, i \neq j$  **do**

$d_{i,j} \leftarrow d(S_i, S_j)$

**if**  $d_{i,j} < d_{min}$  **then**

$d_{min} \leftarrow d_{i,j}$

$S_{i,min} \leftarrow S_i$

$S_{j,min} \leftarrow S_j$

$S \leftarrow S \setminus \{S_{i,min}, S_{j,min}\}$

$S_{i,j} \leftarrow S_{i,min} \cup S_{j,min}$

$S \leftarrow S \cup S_{i,j}$

**until**  $d_{min} > t$ ;

---

The main benefit of this clustering technique is that it does not require prior knowledge of the cluster count, since it is determined automatically using the termination threshold  $t$ . The threshold specifies the maximum allowed Euclidean distance between two clusters. However, since the clusters are given in a high

dimensional descriptor space, the correct choice of  $t$  is not intuitive and furthermore depends on the chosen descriptor type.

After clustering, the created codebook  $C$  contains a list of codebook entries  $\mathbf{c}_j \in \mathbb{R}^n$ , represented as cluster centers from feature descriptors  ${}^f\mathbf{l} \in \mathbb{R}^n$  belonging to corresponding features  ${}^c f_i$ :

$$C = \{\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_k \mid \mathbf{c}_j \in \mathbb{R}^n\} \quad (3.8)$$

### 3.5 Activation

The codebook contains a list of codewords, each of which is prototypical for a specific geometrical property of the training model. However, the codebook does not contain any spatial information yet. If the codebook was used for object detection alone, it might still be possible to accumulate evidence that a given point cloud contains an instance of the object model. Deriving the precise object location will not be possible though, since this task requires additional spatial information for the codebook entries. In accordance with the Implicit Shape Model formulation in [LS03], the *activation* builds a spatial distribution specifying where each codeword can be found on a training model. By iterating again over all training features, the activation matches the features with the codewords contained in the codebook according to an *activation strategy*. This strategy specifies, whether or not a feature can activate a codeword and is based on a distance measure between feature and codeword. Since codewords have been created as the cluster centers from a clustering method applied to features, both are given in the same descriptor space. Thus, the activation strategy can work with the same distance measure as was used by the clustering method during codebook creation.

The simplest method of activation would activate only the best matching codeword for the current feature. However, during codebook creation, a multitude of features has been grouped together to form one codeword. It is obvious that while all features that have been grouped together to create the codeword have a low distance toward each other, as stated by the compactness theorem, there is still noise involved in the correct cluster assignment. In order to allow "fuzziness", the activation allows to use different activation strategies and enabling the activation of more than one codeword.

Given a feature  ${}^c f_i \in F_{tr}$  and the codebook  $C$ , the activation returns those codewords that match the feature according to the strategy, creating the new activation set  ${}^f_i C$ . The distance between feature and codeword is determined by the distance function  $dist({}^f_i \mathbf{l}, \mathbf{c}_j)$ , where  ${}^f_i \mathbf{l}$  represents the descriptor vector associated with feature  ${}^c f_i$  and  $\mathbf{c}_j \in C$  a codeword vector to which the feature is compared against.

- Best codeword

The *Best* activation strategy activates only the best matching codeword to the feature of interest. This is the simplest form of activation, but does not consider noise to the input data.

$${}^{f_i}C^{Best} = \{\mathbf{c} \in C \mid \text{dist}({}^{f_i}\mathbf{l}, \mathbf{c}) = \min(\text{dist}({}^{f_i}\mathbf{l}, \mathbf{c}_1), \text{dist}({}^{f_i}\mathbf{l}, \mathbf{c}_2), \dots, \text{dist}({}^{f_i}\mathbf{l}, \mathbf{c}_k))\} \quad (3.9)$$

- K best codewords

The *KNN* activation strategy activates the  $k$  best matching codewords to the feature of interest. Setting  $k = 1$ , this strategy is equal to the best strategy. However, when activating multiple codewords, uncertainty in the correct choice of correspondence can be compensated.

$${}^{f_i}C^{KNN} = \{\mathbf{c}_j \in C \mid j = 1, 2, \dots, k \wedge \text{dist}({}^{f_i}\mathbf{l}, \mathbf{c}_j) \leq \text{dist}({}^{f_i}\mathbf{l}, \mathbf{c}_{j+1})\} \quad (3.10)$$

- All codewords with a distance below a threshold

The *Threshold* activation strategy depends more on the type of descriptor and the chosen distance measure. It does not put any restrictions on the number of activated codewords, but activates all words whose distance to the feature of interest is below a chosen threshold  $t$ . Obviously, the choice of the threshold depends on the type of the descriptor and the distance measure to compare codewords and descriptors. Without the proper knowledge about the descriptor and the characteristics of the codewords, it is also possible that no codeword will be activated at all, due to an overvalued threshold.

$${}^{f_i}C^{Threshold} = \{\mathbf{c}_j \in C \mid \text{dist}({}^{f_i}\mathbf{l}, \mathbf{c}_j) < t\} \quad (3.11)$$

For each activated codeword  $\mathbf{c}_j \in {}^{f_i}C$  the activation then creates a spatial distribution specifying where each codeword can be found on the object. In this process, each codeword gets assigned references to all the features that activated this codeword. Since features store an absolute position on the object, it is now possible to obtain the locations where a feature activated a codeword. According to the principles of clustering, the current feature provides a geometrical similarity to the activated codeword.

The feature position is still given in absolute coordinates. In order to use the spatial information for object detection, transformation into an object relative coordinate system is inevitable. The object-specific reference position can be obtained in various ways.

Since the object is supposed to be already segmented during training, the object centroid  $\mathbf{x}_c$  can be calculated by:

$$\mathbf{x}_c = \frac{1}{k} \sum_{i=0}^k \mathbf{p}_i \quad \mathbf{p}_i \in P, k = \|P\| \quad (3.12)$$

This centroid represents the *center of mass* of the given point cloud  $P$ . However, another approach became obvious, determining the object position from the bounding box. The *axis aligned bounding box (AABB)* of an object is defined as the object enclosing box and is defined by the minimum and maximum extent of the object in relation to the global reference frame:

$$\mathbf{p}_{min} = \min(P) \quad \forall \mathbf{p}_i \in P : \mathbf{p}_{min} < \mathbf{p}_i \quad (3.13)$$

$$\mathbf{p}_{max} = \max(P) \quad \forall \mathbf{p}_i \in P : \mathbf{p}_{max} > \mathbf{p}_i \quad (3.14)$$

The axis aligned bounding box  $B_{AA}$  is then given by the size  $\mathbf{s}_{AA}$ , the center position  $\mathbf{x}_{AA}$  and the identity rotation  $R_{AA}$ :

$$B_{AA} = \langle \mathbf{s}_{AA}, \mathbf{x}_{AA}, R_{AA} \rangle \quad (3.15)$$

$$\mathbf{s}_{AA} = \mathbf{p}_{max} - \mathbf{p}_{min}$$

$$\mathbf{x}_{AA} = \mathbf{p}_{min} + \frac{\mathbf{s}_{AA}}{2}$$

$$R_{AA} = I^{3 \times 3}$$

The *minimum volume bounding box (MVBB)* is the oriented box enclosing an object, in which rotation, scale and position are computed such that the "empty" space, i.e. the intersection of the object volume with the bounding box volume, is minimal. According to [HP01], computing the MVBB can be reduced to first approximating the diameter of the point set, i.e. finding the maximum distance between a pair of points. The estimated MVBB is then given by the direction between the diameter points and the minimum box enclosing the point set, as described in [BHP01], thus yielding an oriented box  $B_{MV}$ :

$$B_{MV} = \langle \mathbf{s}_{MV}, \mathbf{x}_{MV}, R_{MV} \rangle \quad (3.16)$$

Computing the object's bounding box (either AABB or MVBB) instead of the centroid provides several advantages. Since the object is already segmented prior to training, an object enclosing volume can be computed as mentioned above. Algorithms exist to compute both types of bounding boxes efficiently. Additionally, the bounding box information can be stored with the training data and used at



detection for further analyses. The MVBB provides information about the rotation of an object. Assuming that the bounding box can be reconstructed robustly at detection, comparing bounding boxes gives an insight to how an object has been rotated between scans.

After the bounding box has been computed, the object-specific reference position is given by  $\mathbf{x}_{BB} = \mathbf{x}_{MV}$  as the center position of the minimum volume bounding box. The relative feature position for feature  ${}^c f_i$  can then be given in relation to the object position  $\mathbf{x}_{BB}$  by:

$${}^{f_i} \mathbf{v}_{rel} = \mathbf{x}_{BB} - {}^{f_i} \mathbf{p} \quad (3.17)$$

and represents the vector pointing from the location on the object where the feature was initially computed to the object center position. In order to provide rotation invariance, each feature was associated with a unique and repeatable reference frame given by a rotation matrix  ${}^{f_i} R$ . As described in Section 2.2.3, the local reference frame is computed using local features on the neighborhood of the feature position. It is supposed to be repeatable and rotation invariant. The rotation matrix defining the local reference frame for feature  ${}^c f_i$  is given by  ${}^{f_i} R$  and defines the rotation from the global reference frame into the local feature-specific reference frame. Transforming the vector  ${}^{f_i} \mathbf{v}_{rel}$  from the global into the local reference frame can then be achieved by:

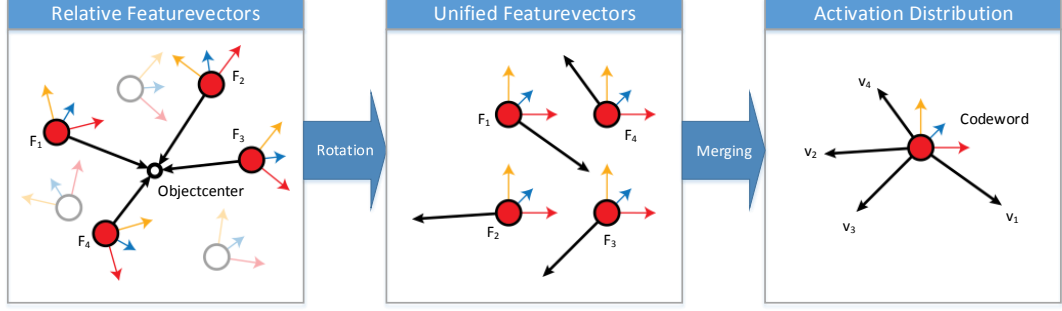
$${}^{f_i} \mathbf{v} = {}^{f_i} R \cdot {}^{f_i} \mathbf{v}_{rel} \quad (3.18)$$

and yielding the final translation:

$${}^{f_i} \mathbf{v} = {}^{f_i} R \cdot (\mathbf{x}_{BB} - {}^{f_i} \mathbf{p}) \quad (3.19)$$

Applying the local reference frame to the feature-relative object position  ${}^{f_i} \mathbf{v}_{rel}$  therefore yields  ${}^{f_i} \mathbf{v}$  and represents the vector from the feature location to the object center in relation to the feature-specific local reference frame, as described by [KPVG10]. Under the assumption that the local reference frame is repeatable both at training and detection,  ${}^{f_i} \mathbf{v}$  provides a position and rotation independent representation for the occurrence of feature  ${}^c f_i$  on the training object for class  $c$ , while the vector  ${}^{f_i} \mathbf{v}$  can then be associated with each codeword that gets activated by  ${}^c f_i$  to form the activation distribution (Figure 3.3).

Since it is possible that individual codewords are not activated by any of the features, a new reduced codebook can be defined which contains only activated codewords.



**Figure 3.3:** Activation procedure during training. Detected features activate a codeword (red) and their relative vectors to the object center are computed. Based on the LRF associated with each of the features, the vectors are rotated into a unified coordinate system. The list of rotated activation vectors then builds the activation distribution for the current codeword.

Codewords that have not been activated can be disregarded, since there will be no spatial distribution computed and they cannot contribute to object detection:

$$\hat{C} = \bigcup_{{}^c f_i \in F_{tr}} f_i C \quad (3.20)$$

However, this only occurs on rare occasions, e.g. when the *Threshold* activation strategy is not able to find matching codewords based on the chosen threshold. Given the activated codewords  $f_i C$  for feature  ${}^c f_i$ , computed from either of the three activation strategies, an inversion process maps each codeword to a list of activated features. Starting with a function  $g$  that maps each feature to its corresponding list of activated codewords  $f_i C$  as described, the inverse function  $g^{-1}$  maps each codeword to its corresponding list of activating features:

$$g : f_i \mapsto \{f_i c_1, f_i c_2, \dots, f_i c_k\} = f_i C \quad (3.21)$$

$$g^{-1} : c_j \mapsto \{c_j f_1, c_j f_2, \dots, c_j f_l\} = c_j F = \{f_i \mid g(f_i) \mapsto f_i C \ni c_j\} \quad (3.22)$$

where  ${}^c_j F$  is the set of all features that activated codeword  $c_j$ . The reversed activation set is then given by:

$$A = \{\langle c_j, {}^c_j F \rangle\} \quad (3.23)$$

Each element of  $A$  maps a list of features  ${}^c_j F$  to a codeword  $c_j$ . Starting with set  $A$ , activation vectors  $f_i v$  are created for each feature  $f_i \in {}^c_j F$  according to Equation (3.19).

The final activation distribution can then be described by:

$$V = \{\langle c_j, {}^{c_j}V \rangle \mid c_j \in \hat{C} \wedge {}^{c_j}V = \{f_i \mathbf{v} \mid \forall f_i \in {}^{c_j}F\}\} \quad (3.24)$$

The set  ${}^{c_j}V$  contains a list of activation vectors for a codeword  $c_j$ , pointing from the feature location on the object to the object center. The activation vectors have, however, already been transformed into the local feature-specific reference frame, as described by Equation (2.3). In combination with the codebook  $\hat{C}$ , the final activation distribution  $V$  thus maps each codeword  $c_j \in \hat{C}$  to its list of activation vectors  ${}^{c_j}V$  and builds the final data pool for the detection process. Algorithm 3 shows the complete activation procedure to build the activation distribution for the codebook. Along with the activation vectors, each entry in the activation distribution also stores additional information like the class  $c$  from the feature that activated the corresponding codeword, and references to the computed bounding box. This additional information is used in the detection process and will be described in the corresponding section.

---

**Algorithm 3:** Activation procedure

---

**Input:** Object center  $\mathbf{x}_{BB} \in \mathbb{R}^3$ , Codebook  $C = \{c_j\}$ , Features  $F_{tr} = \{f_i\}$

**Output:** Set  $V$  of activation vectors

**begin**

$V \leftarrow \emptyset$

**foreach**  ${}^c f_i \in F_{tr}$  **do**

${}^{f_i}C \leftarrow \{c_j \in C \mid {}^c f_i \text{ activates } c_j \text{ according to the activation strategy}\}$

$\hat{C} = \bigcup_{f_i \in F} {}^{f_i}C$

$A \leftarrow \{\langle c_j, {}^{c_j}F \rangle \mid c_j \in \hat{C} \wedge {}^{c_j}F = g^{-1}(c_j)\}$

**foreach**  $\langle c_j, {}^{c_j}F \rangle \in A$  **do**

${}^{c_j}V \leftarrow \emptyset$

**foreach**  $f_i \in {}^{c_j}F$  **do**

$f_i \mathbf{v} = {}^{f_i}R \cdot (\mathbf{x}_{BB} - {}^{f_i}\mathbf{p})$

${}^{c_j}V \leftarrow {}^{c_j}V \cup \{f_i \mathbf{v}\}$

$V \leftarrow V \cup \{\langle c_j, {}^{c_j}V \rangle\}$

---

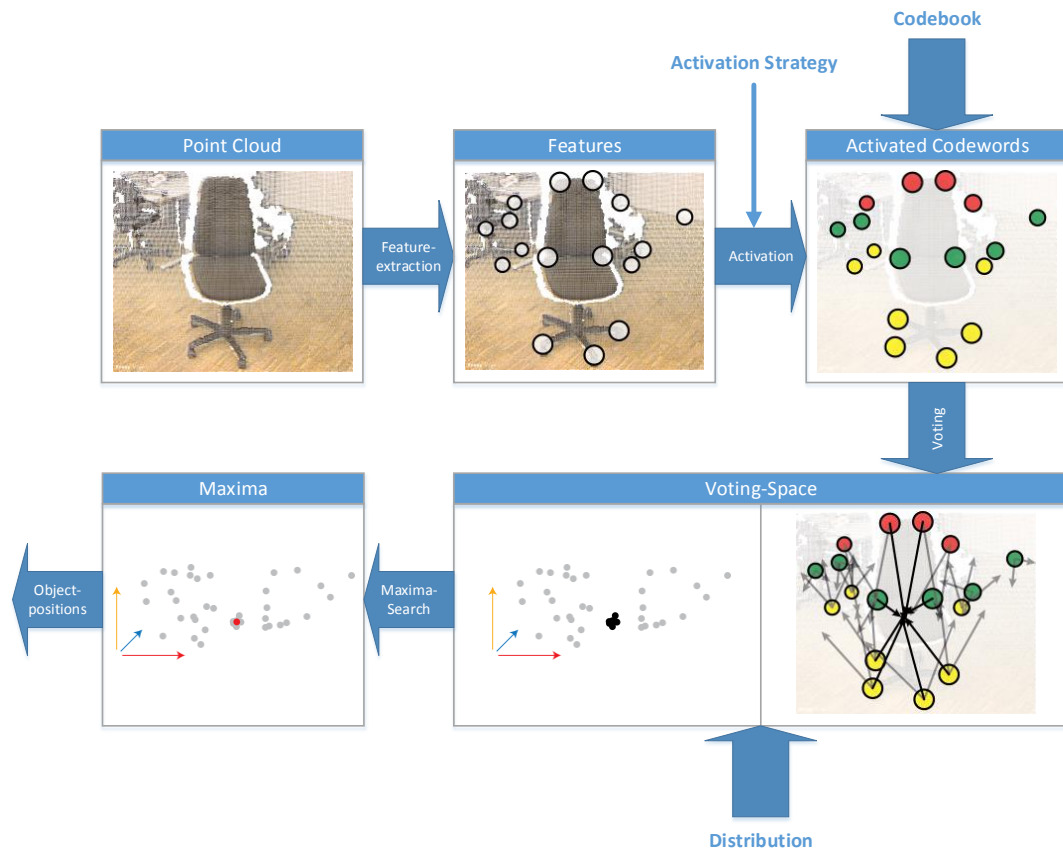


## Chapter 4

# Probabilistic Object Recognition

During recognition, the previously learned implicit 3D representation is applied to yet unclassified input data. Input data is first acquired from an input device. Input acquisition during training aimed at creating appropriate training models representing the training object as a whole. Input data for use in the detection stage poses different requirements to the data acquisition. Since object detection in this stage is supposed to be an automated process, no further analyses should be required. The input device is therefore used to capture a momentary image of the observed real world from a single viewpoint. The captured point cloud is then analyzed for occurrences of object instances by taking into account the previously trained Implicit Shape Model representation.

The training process created an implicit object representation, consisting of a codebook of local geometrical codewords and an activation distribution, reflecting the positions where each codeword is expected to be located on the object. This representation is suited for object recognition, since matching the implicit representation with an unclassified point cloud can reversely provide evidence for the occurrence of an object instance. This is achieved by first detecting features on the input point cloud in identical manner to the training stage. Matching detected features with the previously trained codebook yields a list of activated codewords, that then cast their corresponding activation vectors stored within the activation distribution into a continuous voting space. When an object matches one of the representations stored within the ISM, several activation vectors will create clusters at nearby locations, which represent possible object centers. Since activation vectors have initially been trained as the vectors from a training feature to the object center, the vote clusters are supposed to represent hypothetical bounding box centers of yet unknown object instances. By searching for these maximum density regions, all object hypotheses are collected and evaluated using a weighting strategy. Figure 4.1 presents an overview of the detection pipeline.



**Figure 4.1:** Detection pipeline: Features are extracted on the input point cloud. Detected features are matched with the codebook and matching codewords cast their corresponding activation vectors into a voting space. When an object is found, activation vectors build clusters in the voting space, which are detected by searching for maximum density regions.

## 4.1 Features

Features are detected on the input data just as in training. While the normals had to be oriented consistently among the training object, the input scene used for detection has been captured by a camera from a single viewpoint. Normals can then easily be oriented toward the known viewpoint, as described by Equation (2.2). It is important to note at this point, that although technically possible to do otherwise, keypoints are detected here using the same keypoint detection algorithm as was used in training. Regarding the descriptor, using the same descriptor method is inevitable, since descriptors originating from different methods have

(1) most likely unequal dimensionality prohibiting the comparison of descriptor vectors and (2) they are formulated in a different descriptor space, accounting different geometrical properties and thus are not comparable by design.

In contrast to the list of features  $F_{tr}$  detected during training, the new features  $F_{det}$  represent all detected features on the yet unclassified input point cloud. The chosen descriptor and keypoint detector are supposed to be equal to the methods using for training:

$$F_{det} = \{f_i \mid f_i = \langle^{f_i} \mathbf{p}, ^{f_i} R, ^{f_i} \mathbf{l}\rangle\} \quad (4.1)$$

## 4.2 Activation

After detecting features on the unclassified input point cloud, the activation is performed again similar to Section 3.5 in order to find evidence of the previously trained ISM in the current input data. While the ISM, i.e. codebook and activation distribution, define the possible appearance of an object, the detection stage needs to match input features to the ISM and find correspondences. As a reminder, the codebook contains codewords that define a geometrical property. By matching current features with codewords, correspondences are established at locations where the input data is assumed to match the trained object model.

Given the codebook and the activation distribution, the trained codewords are activated with the input features, yielding a list of activated codewords, which is each associated with a list of activation vectors. The used activation strategy is the same as the one used for training. The activation then yields a number of codewords and their corresponding activation vectors.

According to the chosen activation strategy, the best matches from an input feature  $f_i$  to a number of codewords  $\mathbf{c}_j \in \hat{C}$  is determined. With  $\hat{C}$  the trained codebook and  $f_i$  the feature detected on the yet unclassified input point cloud, the activation yields the best matching codebook entries  $^{f_i} C^{act}$  for feature  $f_i$ , and the total set of activated codewords  $\hat{C}^{act}$ :

$$\hat{C}^{act} = \bigcup_{f_i \in F_{det}} ^{f_i} C^{act} \quad (4.2)$$

In contrast to the activation during training and corresponding Equation (3.20),  $\hat{C}^{act}$  represents the sum of all codewords from the trained ISM that have been activated using the new features  $F_{det}$ . Based on the stored activation distribution, all activation vectors are subsequently collected for each of the activated codewords:

$$\hat{V} = \{\langle \mathbf{c}_j, ^{c_j} V \rangle \mid \forall \mathbf{c}_j \in \hat{C}^{act} \wedge ^{c_j} V = \{\mathbf{v}\} \subset V\} \quad (4.3)$$

### 4.3 Voting

After activation, each codeword casts a number of votes for possible object locations into a multi-dimensional voting space. Each vote contained in the voting space represents an hypothesis for an object occurrence and contributes to the final object hypothesis. While finding the most likely 3D object position is the main goal, the basic hypothesis consists of more than three degrees-of-freedom (DOF). Determining the position, rotation and the specific object class is an 8 DOF problem. While the rotation matrix comprises 9 values, rotations can also be expressed via quaternions with 4 values. Thereby, the 8 degrees of freedom are induced by 3 position coordinates, 4 quaternion elements and the class. Letting each codeword cast votes in an 8 DOF voting space requires a high dimensional maxima search.

However, the voting space can be simplified to three dimensions. Ignoring object rotation at first leaves a four dimensional voting space. Determining the object class can further be accomplished by using heuristics and building up separate voting spaces for each class to be detected. This leaves a three dimensional cartesian space. Voting and object detection is performed in the following way. Given the number of possible object classes from the training stage, individual voting spaces are created for each class. Each activated codeword is further associated with a list of activation vectors, each of which can cast a vote for a specified object class. The codeword then casts all of its votes into the voting spaces, according to the object class of the respective activation vector. Following the voting procedure, the voting spaces are analyzed for maxima individually and in each voting space, maxima are selected according to a probabilistic formulation. Given the final object position and class, rotation is then estimated by collecting all votes that contributed to the object hypothesis and computing a mean rotation over the sum of rotations. Furthermore, reducing the voting space to a three dimensional cartesian space also simplifies the distance measure for the maxima detection method, since the cartesian distance across all three dimensions can be computed in the same way, while computing distances over a seven dimensional non-uniformly scaled voting space needs further adaption and increases complexity.

The voting space  $\mathcal{V}_c$  for a specific class  $c$  is build as a continous three dimensional space, in which every point is assigned an additional weight:

$$\mathcal{V}_c = \{ \langle \mathbf{x}_i, \omega_i \rangle \mid \mathbf{x}_i \in \mathbb{R}^3, \omega_i \in \mathbb{R} \} \quad (4.4)$$

Each activation vector  $\mathbf{v} \in \hat{V}$  that has been collected from activating features with the trained codebook casts a vote for an object hypothesis at a location  $\mathbf{x}_i$ , weighted by  $\omega_i$  into  $\mathcal{V}_c$ , based on the class  $c$  from the training process. The



respective weight corresponds to the probability that this vote reflects the actual object location.

As described in Section 3.5, the activation distribution was created to reflect all observed locations on the training object where the corresponding activated codeword was found. During voting, this process is reversed to backproject the activation vectors from the activated feature locations and indicating possible object center locations.

### 4.3.1 Weighting

Based on the probabilistic formulation in [LS03], the probability for an object identity  $o_n$  at position  $\mathbf{x}$ , given a feature  $f_i$ , can be described by:

$$p(o_n, \mathbf{x}|f_i) = \sum_j p(o_n, \mathbf{x}|\mathbf{c}_j)p(\mathbf{c}_j|f_i) \quad (4.5)$$

Given all codewords  $\mathbf{c}_j$  that match the feature  $f_i$ , the term  $p(o_n, \mathbf{x}|\mathbf{c}_j)$  describes the probability that a codeword supports the object hypothesis based on the activation distribution, while  $p(\mathbf{c}_j|f_i)$  describes the matching probability between codeword and feature. The final probability for the object hypothesis is then given by considering all features  $f_i$  detected on the point cloud:

$$p(o_n, \mathbf{x}) = \sum_i p(o_n, \mathbf{x}|f_i) \quad (4.6)$$

$$= \sum_i \sum_j p(o_n, \mathbf{x}|\mathbf{c}_j)p(\mathbf{c}_j|f_i) \quad (4.7)$$

The probabilistic formulation is implemented using a weighted voting space. Each vote  $\mathbf{v}_i$  being cast therefore includes an additional weight  $\omega_i \in [0, 1]$  that models the probability that this vote supports the correct object hypothesis. Eventually, the maxima represent a region in the voting space with the highest probability density, i.e. all votes around the maximum contribute with their weights to the final probability for the object occurrence.

The final vote weight  $\omega_i$  for vote  $\mathbf{v}_i$  is composed of separate individual weights. The statistical weight  $\omega_i^{st}$  is used to compensate for statistical variations regarding the number of activated codewords and activation vectors per codeword. Since a maximum in the voting space is detected using a number of votes, each vote weight in the maximum's surrounding area contributes to the final probability. The statistical weight therefore guarantees that the probability can be computed independently from the actual number of votes. It normalizes each individual weight based on the number of activated codewords and activation vectors, such

that for a perfect match, in which case each codeword casts at least one vote into the true object center, the maximum probability is 1. Of course, for a different object with a different number of codewords and corresponding activation vectors, the maximum probability is still supposed to be equal to 1, however the individual weights might differ.

Assigning each vote with a normalized weight  $\omega \in [0, 1]$  is problematical. During training, multiple features on an object are detected and clustered into a codebook, and a following activation strategy assigns each codeword a number of votes. During detection, the codewords are again matched against the detected features and the corresponding votes are collected. If only one single feature matches the codebook and is creating a single object hypothesis, it would be considered a high probability even if the multitude of trained codewords has not been matched. Therefore, votes are assigned a statistical weight that aims at compensating statistical differences between the objects and their corresponding implicit shape representations. The statistical weight described in [KPW<sup>+</sup>10] has been used in this thesis in order to normalize the individual votes. The weight  $\omega_i^{st}$  weights a vote for a specified class  $c$ , created from a specified codeword  $\mathbf{c}_j$ , by:

$$\omega_i^{st} = \frac{1}{n_{vw}(c)} \cdot \frac{1}{n_{vot}(\mathbf{c}_j)} \cdot \frac{\frac{n_{vot}(c, \mathbf{c}_j)}{n_{ftr}(c)}}{\sum_{c_k} \frac{n_{vot}(c_k, \mathbf{c}_j)}{n_{ftr}(c_k)}} \quad (4.8)$$

The first term normalizes the vote weight by the number of words  $n_{vw}(c)$  that vote for the specified class  $c$ . The second term normalizes by  $n_{vot}(\mathbf{c}_j)$ , which represents the number of votes assigned with word  $\mathbf{c}_j$ . The third term is used to normalize on inter-class level. It represents the probability that the word  $\mathbf{c}_j$  votes for the given class in contrast to the other classes.

[KPW<sup>+</sup>10] also describes another weight that proved beneficial. The *center weight* is computed for each vote and is already learned during training. When all codewords have been created and the activation distribution has been built, the expected object position is known for each entry. Since the ISM approach allows each codeword to get assigned to a number of votes, not every vote contributes equally well to the final object hypothesis during detection. The weight  $\omega_i^{cen}$  thus computes the distances from each entry in the activation distribution to the actual object center:

$$\omega_i^{cen} = median \left\{ \left( exp \left( - \frac{d(f_i \mathbf{v}, \mathbf{x}_{BB})}{\sigma_{cen}^2} \right) \right) \right\} \quad (4.9)$$

In case a codeword contains a number of votes for a training model, only those votes that actually fall into the surrounding of the known object center position are assigned a high weight. Experiments showed that  $\sigma_{cen}^2 = 0.25$  is a reasonable value.

Additionally, a *matching weight* has been integrated that represents the matching score between a feature and the corresponding activated codeword. A lower matching distance in the descriptor space is supposed to represent a higher probability that the feature actually matches the current codeword, regarding their geometrical similarity. In case feature and codeword represent different visual properties, the activation procedure did not find a better suited codeword for the current feature. Thus, the corresponding matching weight has to be small in order to account for a low matching probability. Given the feature  $f_i$  with descriptor  ${}^{f_i}\mathbf{l}$  and the corresponding matched codeword  $\mathbf{c}_j$ , the matching weight is given by:

$$\omega_i^{mat} = \exp\left(-\frac{\text{dist}({}^{f_i}\mathbf{l}, \mathbf{c}_j)^2}{2\sigma_{mat}^2}\right) \quad (4.10)$$

The right selection of  $\sigma_{mat}^2$  is critical. Since the distance between feature and codeword is given in descriptor space, the value of  $\sigma_{mat}^2$  depends on the chosen descriptor type and indicates how much a codeword can differ from a feature. However,  $\sigma_{mat}^2$  can be estimated during training by the sample covariance. Given each of the features  ${}^cF_{tr} \subset F_{tr}$  on the training model for a specified class  $c$ , the sample mean of distances is computed by:

$$\mu = \frac{1}{MN} \sum_{i=1}^M \sum_{j=1}^M \text{dist}({}^{f_i}\mathbf{l}, \mathbf{c}_j) \quad (4.11)$$

over all features  $f_i \in {}^cF_{tr}$  and activated codewords  $\mathbf{c}_j \in \hat{C}^{act}$ , where  $M = \|{}^cF_{tr}\|$  and  $N = \|\hat{C}^{act}\|$ . The final value of  $\sigma_{mat}^2$  is then computed as the sample covariance:

$$\sigma_{mat}^2 = \frac{1}{MN - 1} \sum_{i=1}^N \sum_{j=1}^M (\text{dist}({}^{f_i}\mathbf{l}, \mathbf{c}_j) - \mu)^2 \quad (4.12)$$

This value is stored during training and computed individually for each class. At detection, the matching weight is evaluated for each vote and  $\sigma_{mat}^2$  is chosen based on the class.

The final weight assigned with each vote combines the individual weights:

$$\omega_i = w_i^{st} \cdot \omega_i^{cen} \cdot \omega_i^{mat} \quad (4.13)$$

### 4.3.2 Rotation Invariance

[KPVG10] introduced three techniques to achieve rotation invariance for hough transform based object recognition methods, as described in Section 2.1.2. While circle voting is a suitable method, experiments showed several disadvantages. In the first stages of object detection, normals are computed on the object. However,

normal computation is still subject to noise and causes disturbances in the normal direction. With circle voting, the radius of the circle depends on the length of the activation vector, i.e. the farther away the activated feature is located from the object center, the bigger the circle will be in the voting space. With small disturbances in normal directions, circles will no longer intersect at precisely determined locations. Additionally, circles need to be subsampled in order to contribute to the voting space. To guarantee equal point distributions independently from feature locations, the sampling density needs to increase with the circle radius. Due to the fact that circle radii depend on the distance of the corresponding feature from the object center, the point concentration in the voting space will increase with the overall object size. Finding the precise maxima in such a dense voting space not only increases computation time but also strengthens false positives. Precise maxima are difficult to extract in an oversampled voting space and circles close to each other promote the creation of irrelevant side maxima.

Since the feature extraction process computes a unique local reference frame at each feature position, rotation invariance can be achieved by assuming a robust and repeatable reference frame. Each activation vector is associated with a corresponding feature on the object and has been rotated from the global reference frame into the feature-specific reference frame during training. Since the LRF only depends on local information around the feature position, the rotation can be reversed during detection by backrotating the activation vector into the global reference frame using the LRF from the current feature.

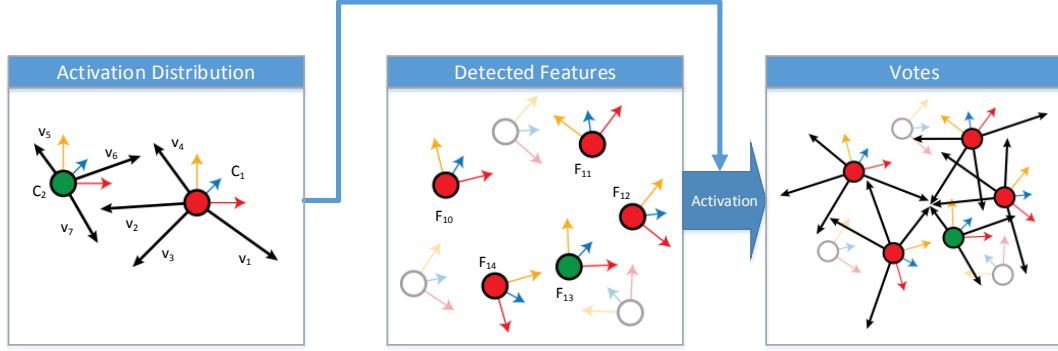
While the activation vector  ${}^{f_i}\mathbf{v}$  has been rotated into the LRF given by the rotation matrix  ${}^{f_i}R$  at training feature  ${}^c f_i \in F_{tr}$ , the rotation procedure during training was given by Equation (3.18):

$${}^{f_i}\mathbf{v} = {}^{f_i}R \cdot {}^{f_i}\mathbf{v}_{rel}$$

During detection, the rotation can be reversed by the inversed LRF matrix  ${}^{f_j}R^{-1} = {}^{f_j}R^T$  computed at a feature position  $f_j \in F_{det}$  on the point cloud, which results in the back-rotated activation vector  ${}^{f_j}\hat{\mathbf{v}}_{rel}$ :

$${}^{f_j}\hat{\mathbf{v}}_{rel} = {}^{f_j}R^T \cdot {}^{f_i}\mathbf{v} \quad (4.14)$$

While the activation vector  ${}^{f_i}\mathbf{v}$  has been created during training, pointing from the feature position to the object center and rotated into the corresponding LRF, the backrotated activation vector for the current feature can now be used to create an object hypothesis.



**Figure 4.2:** Activation procedure during detection. Codewords (red and green) are associated with list of activation vectors during training. Each vector has been rotated into the same coordinate system according to the LRF from the activating training feature. Each detected feature in an unknown scene is associated with its own unique LRF. Features that match the codebook according to the activation procedure then cast the corresponding backrotated votes into the voting space, creating clusters when objects are found.

Given the corresponding feature position, adding the backrotated activation vector to the feature position can be considered a vote for a possible object position  $\mathbf{x}_h$ :

$$\mathbf{x}_h = f_j \mathbf{p} + f_j \hat{\mathbf{v}}_{rel} \quad (4.15)$$

$$= f_j \mathbf{p} + (f_j R^T \cdot f_i \mathbf{v}) \quad (4.16)$$

Since the activation vector was created in relation to the center of the bounding box, the object hypothesis is also the center position of the yet unknown object occurrence and its bounding box (Figure 4.2). Applying rotation and backrotation at training and detection respectively, the final object vote can be considered rotation invariant under the assumption that the LRF computation itself is independent from object rotation.

The voting space is then updated with the computed object hypothesis and the corresponding vote weight:

$${}^e\mathcal{V} = {}^e\mathcal{V} \cup \{\langle \mathbf{x}_h, \omega_h \rangle\} \quad (4.17)$$

### 4.3.3 Hough-Voting

The simplest approach to detecting maximum density regions within the voting space has been adopted from [HA62]. The authors used a voting based approach in order to detect straight lines from images that have been prefiltered with the Canny line detector. Each pixel in this binary image is assumed an occurrence for a possible straight line and therefore casts a number of votes for a possible line equation into a discretized accumulator. Since 2D line equations can be expressed by two parameters, the accumulator array uses two dimensions corresponding to the equation parameters. Starting with an empty array, each vote falls into a specified bin based on their line parameters and increases the accumulator value by 1. Finally, after all possible votes have been cast, the bins are analyzed for maxima using a non-maximum suppression technique. Maxima in the voting space correspond to a detected straight line, defined by the corresponding parameters in the voting space. Backprojecting the parameters from the parameter space into the original image space generates a number of straight lines which correspond to lines in the image.

This approach can be used in a general way to detect arbitrary shapes [Bal81]. In this context, the parameter space models the cartesian space in which the object center is located. The array bins are increased by the corresponding vote weight in order to implement the probabilistic formulation, instead of using an equal weight of 1. Maxima in the voting space can be detected by applying a non-maximum suppression technique and choosing the remaining maximum bins by their accumulator value. Bins with high accumulator values correspond to object locations with a high probability, while low accumulator values can be discarded as irrelevant.

Even though this approach is easy to implement, several disadvantages need to be discussed. The extent of the accumulator bins highly influences the precision of the detected object locations and the performance of the algorithm. In case the bins are too large, two objects close to each other can cast votes into the same bin, in which case the objects are no longer separable. When the bins are too small, noise in the detection process can cause votes for the same object location fall into different bins. A more general issue arises when an object is located close to a bin boundary. Even though the bin size may be chosen appropriately, votes for the same object might fall into different neighboring bins. Even if the maximum probability for the detected object can be considered high enough, it is divided into two different bins. Maxima search might then dismiss the corresponding object, since the individual accumulator values are too low. If an object location was successfully detected within an accumulator bin, the actual position within the bin cannot be precisely determined, thus limiting the precision. Following interpolation can in turn increase precision, but still poses a compromise. Additionally, the

overall size of the accumulator needs to be known in advance, i.e. the range in which detected object locations are expected needs to be known. The larger the accumulator size with constant bin sizes, the higher the computational cost will be. For a smaller accumulator, however, possible object locations are restrained to the overall accumulator extent and objects located outside the accumulator cannot be detected.

#### 4.3.4 Mean-Shift Mode Estimation

The voting space created from the activation can be seen as a sparse representation of a probability density function. While each vote that has been cast into the voting space represents a probability that the object is located at the vote position, finding the actual object location from the multitude of votes in the voting space can be reduced to finding those locations in the probability distribution that have the highest probability density.

In this context, maxima in the probability density functions are detected using the Mean-Shift algorithm described in Section 2.3. Given a point  $\mathbf{x} \in \mathbb{R}^3$ , the mean shift vector applies the kernel profile  $g$  to the point differences between all neighboring points  $\mathbf{x}_i$  within the kernel bandwidth  $h$ . Since the Mean-Shift Mode Estimation now searches for maxima in the voting space, the data points represent the individual votes, i.e.  $\mathbf{x}_i \in {}^c\mathcal{V}$ . The initial mean shift vector was defined by Equation (2.30):

$$\mathbf{m}_{h,G}(\mathbf{x}) = \frac{\sum_{i=1}^n \mathbf{x}_i g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)}{\sum_{i=1}^n g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)} - \mathbf{x}$$

This equation describes the mean shift vector as the mean of the differences between all points within the bandwidth acquired with a specified kernel. The voting space, however, contains a set of weighted votes. Instead of computing the mean shift over uniformly weighted data points, a modification to the mean shift vector, as proposed in [Che95], applies the kernel density gradient estimation over a weighted point set, in which each data point  $\mathbf{x}_i$  is assigned with a weight  $\omega_i \in \mathbb{R}$ :

$$\hat{\mathbf{m}}_{h,G}(\mathbf{x}) = \frac{\sum_{i=1}^n \mathbf{x}_i \omega_i g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)}{\sum_{i=1}^n \omega_i g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)} - \mathbf{x} \quad (4.18)$$

The trajectory for the mean shift algorithm, given a weighted point set, is therefore given by modifying Equation (2.31):

$$\mathbf{x}^{(t)} = \frac{\sum_{i=1}^n \mathbf{x}_i \omega_i g\left(\left\|\frac{\mathbf{x}^{(t-1)} - \mathbf{x}_i}{h}\right\|^2\right)}{\sum_{i=1}^n \omega_i g\left(\left\|\frac{\mathbf{x}^{(t-1)} - \mathbf{x}_i}{h}\right\|^2\right)} \quad (4.19)$$

The standard mean shift approach would begin iteration from each of the individual data points as a start points, also called *seed points*. Since a maximum density region is naturally composed of multiple data points, several seed points would converge to the same maximum. With increasing number of data points and even with parallelization, the algorithm would soon require much processing power in order to compute only few maxima. In order to improve performance, a regular grid of appropriate grid size is superimposed on the data points before applying mean shift. Seed points are then sampled from the grid cells, whereupon each cell containing at least a specified number of data points creates a seed point. The precise creation of the seed point is negligible. Possibilities include the grid cell center, using one of the included data points or the centroid of included points. In this approach, seed points are created at the cell intersections. Further attention should be paid to the selection of the appropriate grid size. The maximum number of detected maximum density regions cannot be higher than the number of seed points. Thus, an undersized grid size may cause the algorithm to miss important maxima. In this thesis, the grid size was chosen as  $2^{(-0.5)}2h$ , since the corresponding cell is then perfectly covered by the kernel. This parameter choice has proven reasonable in experiments. Since the seed points converge to a stationary point independently from each other, it is likely that two or more seed points converged to approximately the same position. In order to retrieve individual maxima, a following pruning step performs a non-maximum suppression technique and eliminates duplicate maxima. The final Mean-Shift Mode Estimation is depicted in Algorithm 4.

Additionally, the final probability for the detected maximum  $\mathbf{x}^{(t)}$  is given by the kernel density estimation at the maximum position in the voting space. Since the voting space consists of weighted data samples, the kernel density estimator from Equation (2.26) is modified to incorporate the sample weights  $\omega_i \in \mathbb{R}$ :

$$p(\mathbf{x}^{(t)}) = \sum_{i=1}^n \omega_i k\left(\left\|\frac{\mathbf{x}^{(t)} - \mathbf{x}_i}{h}\right\|^2\right) \quad (4.20)$$

A object hypothesis can then be characterized by the hypothesis position and its corresponding probability:

$$o_i = \langle \mathbf{x}^{(t)}, p(\mathbf{x}^{(t)}) \rangle \quad (4.21)$$



**Algorithm 4:** Mean-Shift Mode Estimation**Input:** Voting space  ${}^cV = \{\langle \mathbf{x}_1, \omega_1 \rangle, \langle \mathbf{x}_2, \omega_2 \rangle, \dots, \langle \mathbf{x}_n, \omega_n \rangle\}$ , Threshold  $t$ **Output:** Set  $Y$  of maxima**begin** $Y_t \leftarrow \emptyset$  $S \leftarrow \{\mathbf{s}_j \mid \mathbf{s}_j \text{ are seed points created from } {}^cV\}$ **foreach**  $\mathbf{s}_j \in S$  **do** $\mathbf{x}^{(t)} \leftarrow \mathbf{s}_j$ **repeat**

$$\mathbf{m}^{(t)} \leftarrow \frac{\sum_{i=1}^n \mathbf{x}_i \omega_i g\left(\left\|\frac{\mathbf{x}^{(t-1)} - \mathbf{x}_i}{h}\right\|^2\right)}{\sum_{i=1}^n \omega_i g\left(\left\|\frac{\mathbf{x}^{(t-1)} - \mathbf{x}_i}{h}\right\|^2\right)} - \mathbf{x}^{(t-1)}$$

$$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} + \mathbf{m}^{(t)}$$

**until**  $\|\mathbf{m}_{h,G}(\mathbf{s}_j)\| < t$ ; $Y_t \leftarrow Y_t \cup \{\mathbf{x}^{(t)}\}$  $Y \leftarrow \{\mathbf{x}^{(t)} \mid \mathbf{x}^{(t)} \text{ has no side maxima in its neighborhood}\}$ 

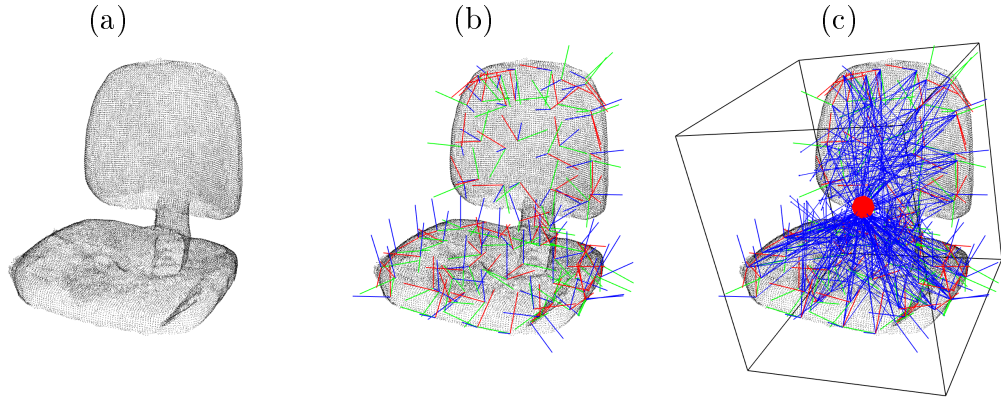
## 4.4 Multi-Class Detection

As addressed in Section 4.3, the high dimensional voting space is reduced to three dimensions, in which a location in the voting space corresponds to a possible object position. In order to detect object occurrences in multiple classes, separate voting spaces are constructed for each class  $c$ , denoted by  ${}^cV$ . Maxima are then detected in each voting space individually, while the maxima are merged between the voting spaces. A user-defined probability threshold specifies, when an object hypothesis is considered valid, i.e.  $p(\mathbf{x}^{(t)}) > t_p$ . The final set of object hypotheses is then given by

$$O = \{o_i \mid o_i = \langle \mathbf{x}^{(t)}, p(\mathbf{x}^{(t)}) \rangle \wedge p(\mathbf{x}^{(t)}) > t\} \quad (4.22)$$

## 4.5 Bounding Box

During training, each training object has been associated with a minimum volume bounding box. After detection, the bounding box can be extracted from the votes that contributed to the object hypothesis. During the training procedure, the bounding box information has been stored as additional information for each computed vote. Since a vote is generated from the activation strategy by matching a feature to a codeword, the corresponding feature is associated with a local reference frame. In order to achieve rotation invariance, the bounding box orien-



**Table 4.1:** Detection process and bounding box extraction. (a) The original object that is to be detected. (b) Computed features and their corresponding local reference frames. (c) Features are matched with the codebook and activated features cast votes into the voting space. The Mean-Shift Mode Estimation extracted the predicted object location and the bounding box has been averaged from the sum of contributing votes.

tation that is initially given in the global coordinate system is transferred into the feature-specific coordinate system using the local reference frame. Aside from creating a vote, each feature that activates a codeword also stores the feature-relative bounding box for the created vote.

This information can then be used during detection. All detected features on the unclassified point cloud that activate a codeword cause the activation strategy to cast a number of votes into the voting space. During training, each vote has been assigned a feature-relative bounding box. When casting the votes, the associated bounding boxes are transferred back into the global coordinate system using the corresponding local reference frame for the current feature. After the maxima detection has detected the most likely object hypotheses, all votes are collected that contributed to the hypothesis. Since the Mean-Shift Mode Estimation estimated the kernel density at the detected maxima location, all votes within the kernel bandwidth are collected (Figure 4.1). This results in a list of bounding box hypotheses, each of which has been weighted according to the corresponding vote weight.

Estimation of the bounding box is then performed by creating an average bounding box representation on the basis of the collected weighted votes, enforcing the constraint  $\sum \omega_i = 1$ . While averaging the bounding box size can easily be done, computing an average weighted rotation is nontrivial. Averaging the orthogonal basis vectors of the rotation matrix discards the orthogonality of the matrix. Instead, the rotation matrix is converted into a quaternion representation. Averaging quaternions can then be achieved by computing the  $4 \times 4$  scatter Matrix

$$M = \sum_{i=1}^N \omega_i \mathbf{q}_i \mathbf{q}_i^T \quad (4.23)$$

over all quaternions  $\mathbf{q}_i$  and their associated weights  $\omega_i$ . After computing the eigenvalues and eigenvectors of  $M$ , the eigenvector with the highest eigenvalue corresponds to the weighted average quaternion [MCCO07].



# Chapter 5

## Evaluation

In this chapter, the developed algorithm is analyzed with respect to performance and precision and applied to several datasets and test cases.

At first, the used datasets and their methods of creation are described in Section 5.1. Since the presented method relies on a huge number of different parameters, it is not possible to evaluate the detection results of all parameter permutations. Instead, the best choice of parameters is analyzed methodically in Section 5.2. After the best parameters have been found, the precision of the algorithm is determined using two use cases. In Section 5.3, the trained algorithm is used to classify single object point clouds into the trained classes under the assumption that the available point clouds contain only one isolated object of one class. In Section 5.4, the algorithm is used to detect object occurrences in unclassified point clouds from two datasets. The input point clouds can contain multiple objects of multiple classes in a scene. The performance of the algorithm is evaluated in Section 5.5. Finally, the results are discussed in Section 5.6.

### 5.1 Datasets

Evaluation is performed on two different datasets. The first dataset was created using depth images created with a Kinect camera. The second dataset was taken from a publicly available dataset and adapted to the relevant evaluation test cases. The following section describes the characteristics of the individual datasets and reviews how the data was generated.

#### 5.1.1 Kinect Dataset

In a first attempt, Kinect Fusion was used to create training models. This algorithm is specifically adapted for use with the Microsoft Kinect camera and performs

a real-time registration of the currently captured point cloud with a global model by executing its algorithms on the graphics processor (GPU). This allows the user to move around with the camera and build a complete world model instantly. In order to scan in realtime, a state-of-the-art GPU is beneficial. However, comparing the scanned objects with scenes captured from a single viewpoint shows that Kinect Fusion does not precisely retain the object dimensions. During the surface reconstruction, the surface is smoothed, causing the scanned object to shrink. Since the presented algorithm is not scale invariant, varying object dimensions would cause the votes to scatter around in the voting space and hindering maxima detection. Training objects were therefore captured from different angles and the point clouds were combined into a complete training model, as described in the following section. The presented dataset is referred to as *Kinect Dataset* in the following sections.

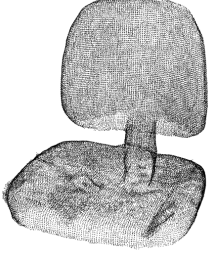
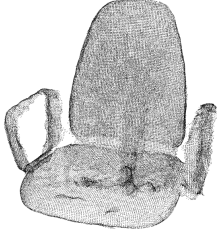
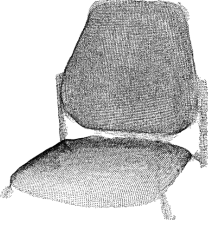
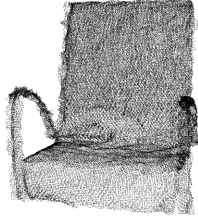
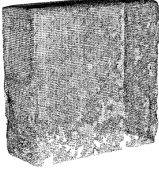
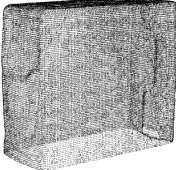
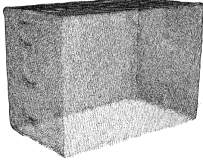

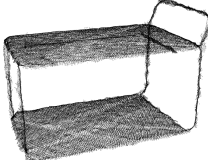
## Models

For each training model, several distinct scans from different viewpoints have been captured using the Kinect camera. Automatic registration using ICP was rejected, since no initial pose estimates were available. The different point clouds for the objects were therefore aligned manually to create the final training model. Applying ICP after the manual alignment is possible. However, the results were already acceptable and the object dimensions were appropriate. The aligned point clouds were merged and the resulting point cloud was sampled with a uniform grid to create a uniform point distribution. Finally, the point cloud was smoothed with *Moving Least Squares* [ABCO<sup>+</sup>03] in order to compensate for noise on the object surface and filtered using a statistical outlier removal method provided by the PCL.

Table 5.1 shows the classes and their training models, along with their associated names. Please note that the illustrations are not true to scale.

Attention was paid to creating models that contain only parts that represent the object appropriately. Especially for class containing chairs, the models do not include the chair legs or the office chair bogies. These structures are typically thin and shiny, which often leads to misreadings in the sensor data and can distort detection results. In case of the office chair, the bottom part of the original chairs is allowed to spin around one axis. Objects that are able to change their shape can lead to missed detections, since votes in the voting space originating from the deformable parts will be scattered around. Training objects therefore need to be fixed in size and shape and a rotating office chair bogie would cause the algorithm to loose precision.

**Table 5.1:** Training models from Kinect dataset.

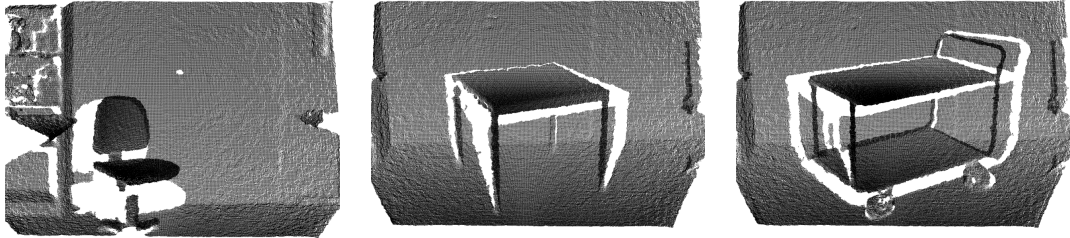
Class 0 (Chairs):			
chair1	chair2	chair3	chair4
			
Class 1 (Computers):			
comp1	comp2		
			
Class 2 (Tables):			
table1	table2	table3	
			

## Scenes

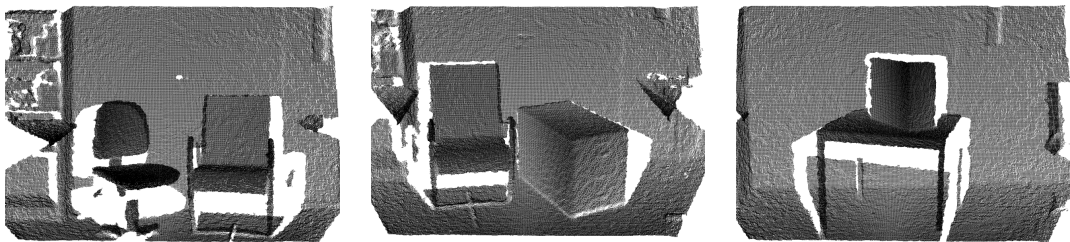
Several scenes have been captured that aim at representing the most probable use cases. They have been captured using a Kinect camera and stored in full resolution. Afterwards, normals have been computed on the point cloud data and have been oriented toward the camera viewpoint. The normal information was stored along with the point cloud data to improve the automatic evaluation process and avoid re-computing normals on every test run. The scenes are split into three categories, containing either single objects, multiple objects or occluded objects. Table 5.2 shows a representative excerpt of the used test scenes.

**Table 5.2:** Sample scenes from the Kinect dataset with their corresponding classes.

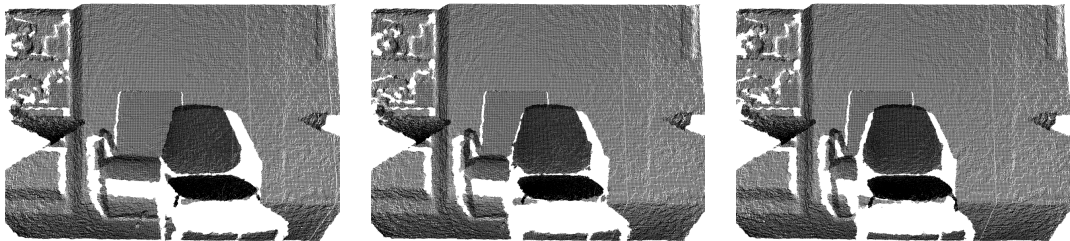
## Category 1: Single Objects



## Category 2: Multiple Objects

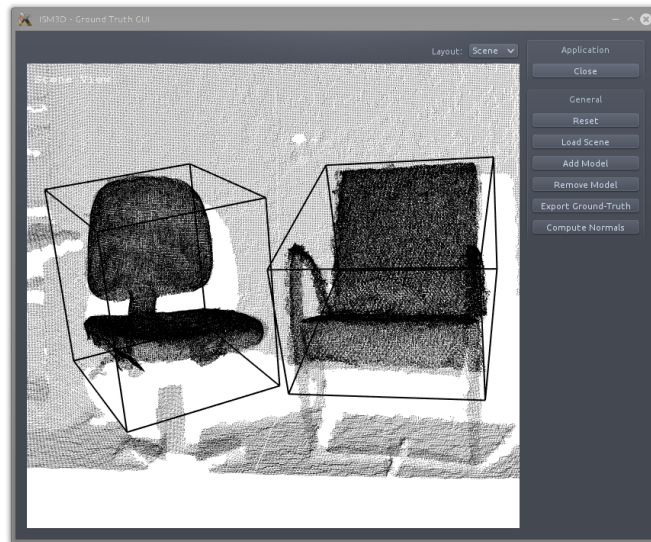


## Category 3: Occlusions

**Ground Truth**

Ground truth data has been created by manually aligning training models in each of the scenes to the object that is supposed to be detected. The exported ground truth data contains the center position of the bounding box along with the name of the aligned model. In the experiments, the detected objects are matched with the ground truth data for the analyzed scene. Figure 5.1 shows the developed application for annotating point clouds with ground truth data.





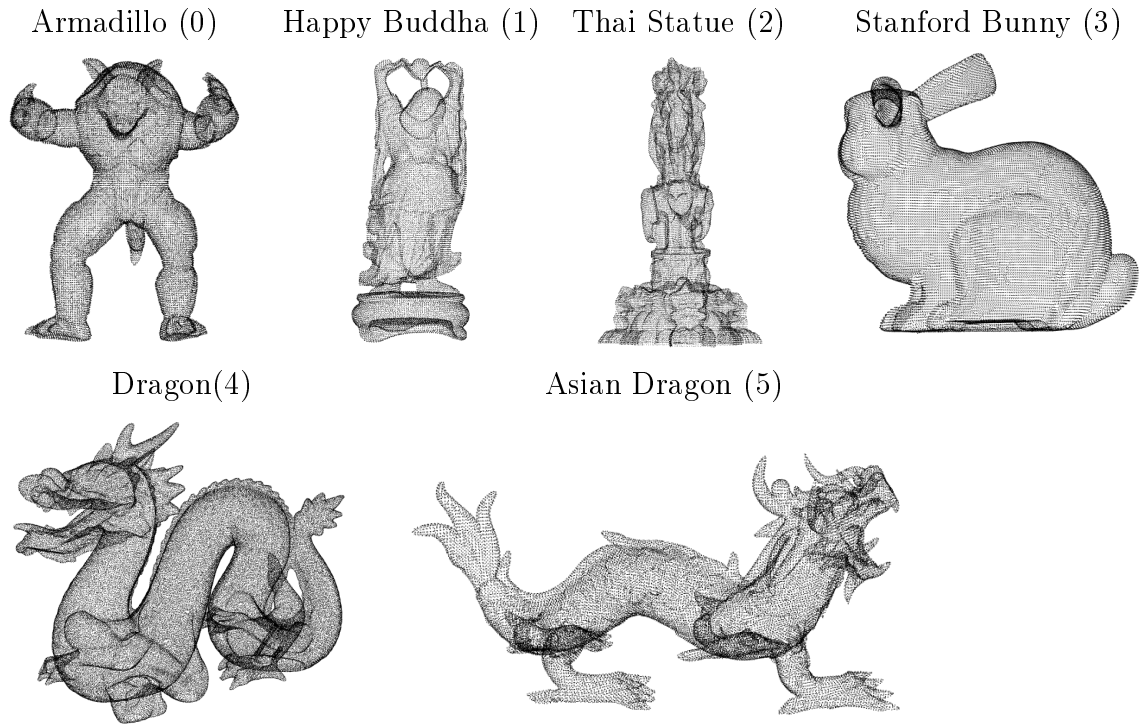
**Figure 5.1:** Ground truth annotation on a point cloud. The two objects have already been annotated and their bounding boxes are displayed.

### 5.1.2 Stanford Dataset

In order to perform evaluation also on external datasets, models from the *Stanford 3D Scanning Repository* have been used [Sta14]. [TSDS10] and [TSDS11] used an adaptation to the Stanford dataset for evaluation of the SHOT descriptor [Com14] (*Dataset 1 & 2 (Stanford)*). They used the original models to create artificial scenes containing 3 - 5 different models, rotated and translated randomly. Additionally, it has been paid attention to the fact that the individual models in the scenes do not intersect. For this thesis, this approach has been adapted. While the scenes are already available from the described SHOT dataset, the models as well as the scenes do not contain normal information. The models from the Stanford dataset have thus been subsampled to achieve an identical point cloud resolution for each model, followed by computing consistently oriented normals using the approach described in Section 2.2. The preprocessed training models have then been rotated and translated using the ground truth data contained in [Com14], resulting in the same dataset with additional normal information.

#### Models

The training models have been created from the Stanford 3D Scanning Repository [Sta14] and contain a total of 6 different models, as seen in Table 5.3. For evaluation, each of the different models will be treated as an individual class, indicated by the number behind the model name.

**Table 5.3:** Training models from the Stanford dataset.

### Scenes

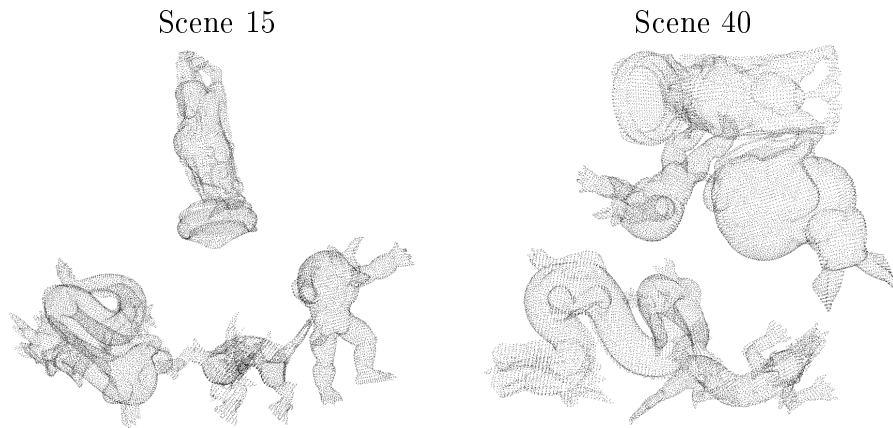
The previous collection of training models from the Stanford 3D Scanning Repository has been used to create artificial scenes containing different permutations of the models, translated and rotated randomly. The test cases consist of  $3 \times 15 = 45$  different scenes with each 3 – 5 different models (Table 5.4).

### Ground Truth

The ground truth is provided along with the dataset as a transformation matrix that specifies the transformation from the corresponding model to the model representation in the scene. This data is used in the evaluation process to verify the quality of the detection.

## 5.2 Parameter Selection

Before analyzing the overall precision of the algorithm in several test cases, the effects of parameter selection are evaluated. The algorithm depends on a variety

**Table 5.4:** Sample scenes from the Stanford dataset with their corresponding classes.

of different parameters, and their precise effects are not always obvious. Especially the choice of the different feature detection components and the process of codebook creation are of interest and will be evaluated in the following section.

### 5.2.1 Features

In order to analyze the right selection of keypoint detectors and descriptors, the possible combinations have been analyzed and evaluated for detection precision. The evaluated keypoints are:

- **Harris3D**  
The Harris3D interest point detector is implemented in the PCL and detects salient points on the surface of an object by analyzing changes in normal directions on the local neighborhood.
- **Intrinsic Shape Signatures**  
The ISS interest point detector analyzes the characteristics of the covariance matrix of each point neighborhood and creates interest points where the neighborhood shows large point variations.
- **VoxelGrid**  
The voxel grid is a regular grid with a specified cell length and is superimposed on the objects. Keypoints are sampled regularly on the object as the centroids of points within the voxel grid cell.

The Harris3D and the ISS interest point detector create keypoints at precisely determined locations on the object surface by analyzing the local point neighborhoods. They comply to the keypoint definitions as described in Section 2.2.2. In

contrast, using a voxel grid aims at creating a large amount of keypoints instead of precisely located points. The idea is that with a large number of keypoints, even though the point characteristics do not precisely match other keypoints, correspondences can still be established if enough keypoints are sampled around a region of interest. The underlying assumption is that the probability of a correct point correspondence is high if only enough points are sampled on the object surface.

For the given evaluation, the ISM has been trained with one model only and the detection algorithm has been applied to a scene containing a single instance of the trained model. Depending on the selected descriptor and keypoints detector, the particular object hypothesis has been selected among the list of detected objects that has the minimum distance to the ground truth. The results of the evaluation are depicted in Table 5.5.

**Table 5.5:** Minimum distance between ground truth and detected objects. All measurements are given in meters.

	PFH	FPFH	SHOT
Harris3D	0.0732	0.1915	0.3907
ISS	0.0719	0.0674	0.0664
VoxelGrid	0.0751	0.0556	0.0623

One can clearly see that while the Harris interest point detector does not work well with the FPFH and the SHOT descriptor, the other combinations lead to a detected object hypothesis whose distance to the ground truth is acceptable. For this context, an object is considered successfully detected if the distance to the ground truth does not exceed  $0.1 m$ .

However, the previous analysis does not allow any conclusions to the significance of the object hypothesis. It only states that the algorithm was able to detect an object at the expected location. During detection, the algorithm detects maxima in the voting space and computes a probability for each detected object hypothesis, based on the total weights of all contributing votes. The detection result is represented by a list containing each object hypothesis, sorted by the computed probability. A measure for the significance of the hypothesis is therefore the rank of the hypothesis within the sorted list of detected objects. The results are shown in Table 5.6.

Apparently, the Harris interest point detector works poorly with any of the available descriptor types. Among all detected objects, the object hypothesis with the minimum Euclidean distance to the ground truth is detected at positions  $> 50$  in the detection list. Looking at Table 5.5 shows that for these hypotheses, the Euclidean distance itself is not acceptable for the current task. While the ISS keypoint detector produces more reasonable results, only the voxel grid in combi-

**Table 5.6:** Index of the minimum distance hypothesis.

	PFH	FPFH	SHOT
Harris3D	67	51	69
ISS	11	7	11
VoxelGrid	1	0	0

nation with either FPFH or SHOT descriptor generates an object hypothesis at the topmost position in the detection list. These results are intensified by Table 5.7, which depicts the distance between the ground truth and the first object hypothesis, i.e. the topmost hypothesis in the detection list. Only the voxel grid based keypoint detector in combination with the FPFH or SHOT descriptor leads to a detected object that is within the acceptable margin.

**Table 5.7:** Distance between ground truth and first object hypothesis. All measurements are given in meters.

	PFH	FPFH	SHOT
Harris3D	1.289	1.585	1.597
ISS	0.882	1.189	1.174
VoxelGrid	1.116	0.056	0.062

These variations have several causes. With the Harris3D and the ISS interest point detector, only few sparse keypoints are detected on salient points on the object surface. Depending on the selected clustering method and activation strategy, each of the detected codewords is assigned only a few votes, based on the matching features. During detection, these votes are cast into the voting space. With the chosen keypoint detector, the following detection algorithm is highly sensitive to noise, since small variations in the data naturally have effects on the detected keypoints. The process of matching codewords to keypoints therefore also suffers from influence to noise, which causes potential maxima to scatter around in the voting space. Since these maxima typically are not composed from lots of votes, these small variations caused by noise lead to imprecision in the Mean-Shift algorithm. In contrast to common keypoint detectors, the voxel grid approach takes its advantages only from the vast amount of keypoints, causing maxima in the voting space to be composed from lots of votes, therefore stabilizing detected maxima.

Following these results, the voxel grid based keypoint detector is chosen for the following evaluation process. While both FPFH and SHOT are to be considered as descriptor, SHOT was chosen over FPFH for its runtime performance. FPFH already poses a runtime enhanced version of the slow PFH descriptor. However,

SHOT still requires about 30 times less computation time compared to FPFH on a current state-of-the-art computer.

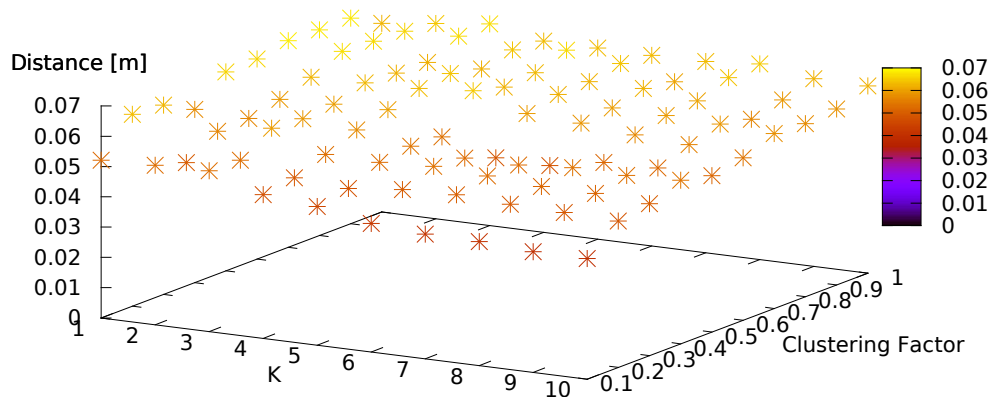
### 5.2.2 Codebook Creation

One interesting aspect is the relationship between the codebook creation process using clustering, and the activation strategy. The clustering groups together features among all training models and classes and creates geometric codewords. The activation strategy on the other hand defines how the created codewords and the detected features are combined to create the implicit shape representation and how the votes are created for an unknown scene.

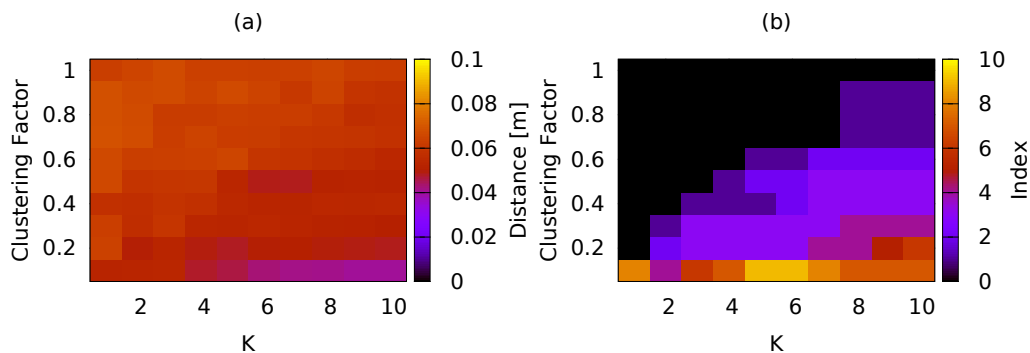
In order to visualize the relationship between the parameters, training and detection were performed multiple times with different parameters and test cases and evaluated for precision compared to ground truth data. Training was performed using the *KNN activation strategy* and *K-Means clustering* on a number of training models and class combinations. For the clustering, the number of clusters and therefore codewords was iteratively changed from 100 % to 10 % of the detected number of features, with a step size of 10 %. At the same time, the number of activated codewords per feature was changed from 1 to 10. Using the multitude of trained Implicit Shape Model representations as described, a scene was analyzed for object occurrences. The detection process yields a list of detected object hypotheses, sorted by the total weight for the current hypothesis, while each entry is furthermore attached with a class and the object position. From the computed object hypotheses, the object position with the minimum distance to the ground truth is collected, as well as the index in the list and the total weight. Over each test case, this information is collected as a function of the two altered parameters and converted into a plot, as shown in Figure 5.2.

This graph shows the detection precision for an Implicit Shape Model trained with a single object (chair1) and the detection applied on a scene containing only one instance of the corresponding object (scene3). It shows how the detection precision differs with the parameter choice. The x-axis depicts the chosen parameter K for the activation strategy, while the y-axis represents the number of created clusters by the clustering factor, that determines the number of created codewords from a factor multiplied by the number of detected features. It is important to note that the complete dependency is a function of two parameters and is therefore best represented using a three dimensional graph.

Given the ground truth data, the object hypothesis with the minimum Euclidean distance to the ground truth is retrieved and the distance is plotted on the z-axis as the height of the graph points. In order to improve visualization, the color coding also represents the distance. It can clearly be seen that for each combination of K and the cluster count, the minimum distance between the ground



**Figure 5.2:** Detection precision graph (training chair1, detection scene03): Minimum distance to ground truth in relationship to the codebook size and the activation strategy.



**Figure 5.3:** Detection precision map (training chair1, detection scene03): (a) Minimum distance to ground truth in relationship to the codebook size and the activation strategy. (b) Corresponding index in the detection list for the object hypothesis with minimum distance to the ground truth.

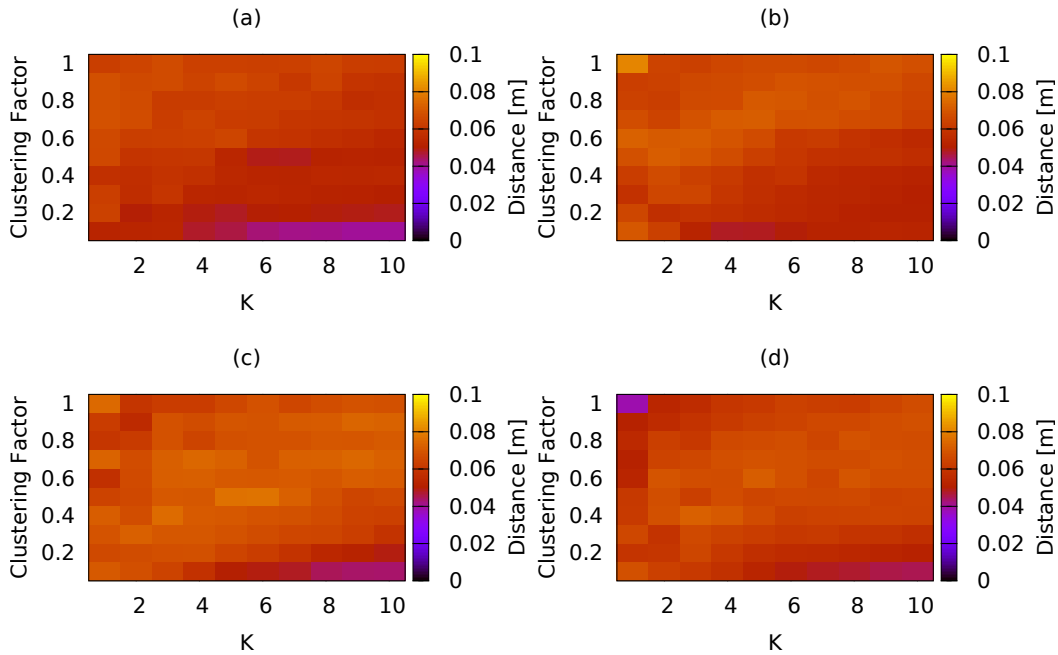
truth and the detected object hypotheses is always within the specified margin of  $0.1 m$ . This means, that in each case, at least one object hypothesis is located at the ground truth position for the current scene.

Visualization can also be achieved by representing the function as a map, as shown in Figure 5.3, which is more suitable in the given context. Figures 5.2 and 5.3 (a) both show a slightly improving precision with increasing values for  $K$  and decreasing values for the cluster count.

However, not only the minimum distance between ground truth and the detected objects is of interest. The detection component creates a sorted list of hypotheses, while the detected object with index  $i$  in the list has been assigned a higher total weight than the object with index  $i + 1$ . Therefore, the index can be considered a direct measure for the certainty of the detection, as shown in Figure 5.3 (b). The black area depicts the cases in which the object with minimum distance to the ground truth is also the topmost entry in the detection list, therefore the most significant object detection. While the previous data suggested that the precision improved with increasing  $K$  and decreasing cluster count, this data suggests that in these cases, the object significance decreases. This can be explained by the characteristics of the clustering method and the activation strategy. With increasing  $K$ , each feature activates the  $K$  best matching codewords and therefore creates lots of votes for the corresponding codewords. With decreasing cluster count, more features are grouped together to create a codeword, therefore broadening the generalization. Whenever a feature activates a codeword at detection, this codeword was initially created from a large number of training features. This also widens the range in which a feature is allowed to activate a codeword, as predicted from the compactness theorem. The clustering creates a compact representation, while the individual clusters are most distinct. Therefore, the number of votes will also increase for a decreasing number of clusters. Thus, both parameters affect the number of votes created. However, the number of votes and therefore the vote density has influence on the precision, since the voting space might be dense in such a way that lots of maxima are detected at insignificant locations. The improving precision with increasing parameters therefore does not necessarily correlate with actual object recognitions, but rather with an oversampled voting space and false positives. This theory is strengthened by Figure 5.3 (b), which shows that for the previously discussed cases, the index for the object hypothesis with minimum distance to the ground truth rises, therefore indicating insignificant detections.

The same test cases were conducted multiple times with different configurations of training objects and classes, as seen in Figure 5.4. It can clearly be seen that the increasing number of training models does not have much effect on the minimum object distance to the ground truth. The distance is always in an acceptable range of  $0.1m$  from the ground truth. However, while the previous graphs showed the correlation between the clustering and activation parameters for the detected object hypothesis with the minimum distance to the ground truth, Figure 5.5 depicts the ground truth distance for the first detected object. Since the algorithm creates a sorted list of hypotheses, the first detected hypothesis is supposed to represent the most likely object detection. The rank in the detection list is determined by

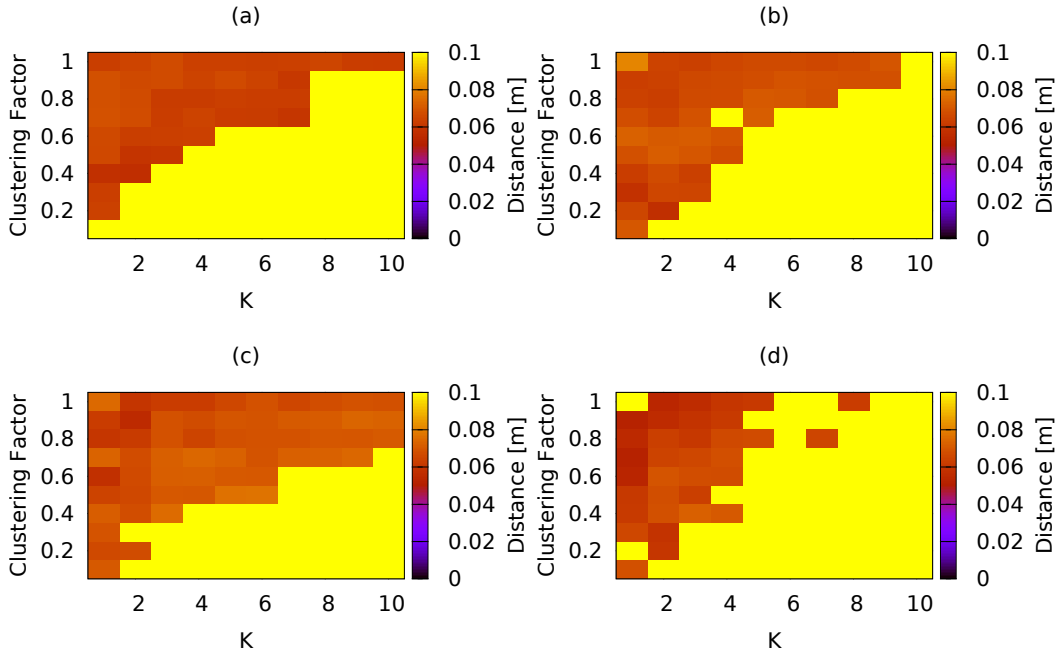




**Figure 5.4:** Detection precision on the object hypothesis with minimum distance to the ground truth on scene3. (a) Training chair1. (b) Training chair1 and chair2. (c) Training chair1, chair2 and chair3. (d) Training chair1, chair2, chair3 and chair4.

the vote weighting, that is created using statistical analyzes and is supposed to weaken unlikely and strengthen likely hypotheses.

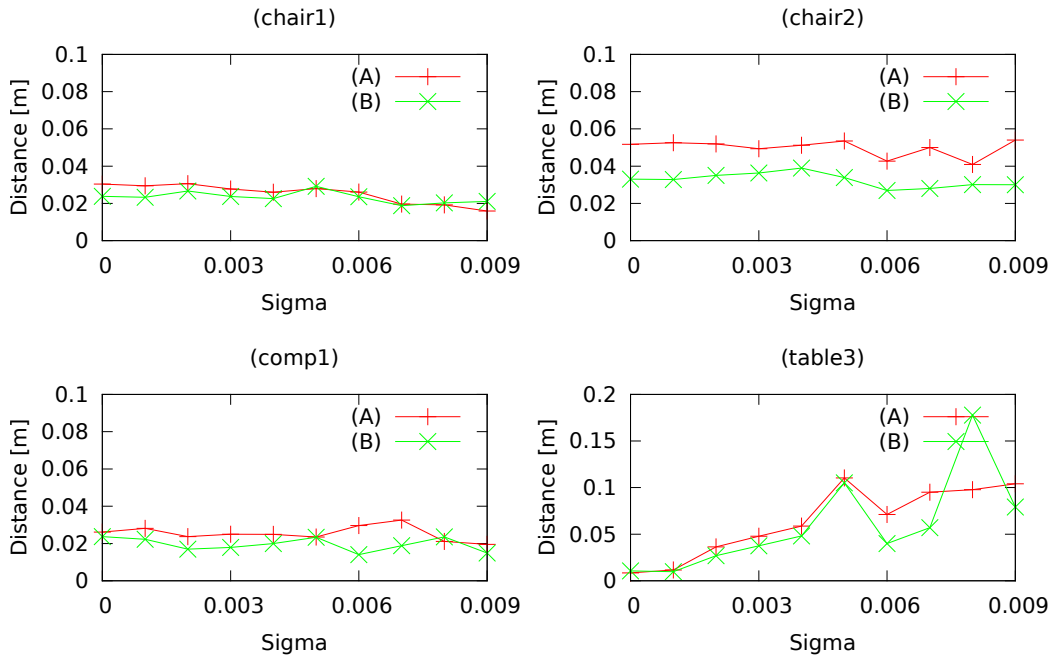
The graph shows that for the first detected hypothesis, the distance to the ground truth is acceptable in most cases. An outlier can be noticed for Figure 5.5 (d), in a case where the distance of the first object hypothesis has been computed as  $1.584m$ . Not shown in the graph is the second hypothesis, which has been detected at an acceptable distance of  $0.038m$  to the ground truth. These results indicate that the best detection precision is achieved in ranges where the activation strategy activates only a small number of codewords, while using a codebook with only little or no clustering at all. The main reason for using Implicit Shape Models, i.e. creating a codebook of geometric words, is therefore not applicable for use with 3D data. Instead, the results confirm the implications of [STDS10], that states that the codebook size is not expected to have any positive influence on the detection capabilities of the algorithm, but are rather legitimated by performance considerations.



**Figure 5.5:** Detection precision on the first object hypothesis on scene3. (a) Training chair1. (b) Training chair1 and chair2. (c) Training chair1, chair2 and chair3. (d) Training chair1, chair2, chair3 and chair4.

### 5.3 Classification

The classification task evaluates the ability of the algorithm to detect the true object class, given a point cloud containing only the segmented object of the corresponding class. The input point cloud therefore does not contain any data besides the well-segmented object. In the presented test cases, the training objects from the Kinect dataset have been rotated and translated randomly and Gaussian noise was applied with increasing values for sigma. The Implicit Shape Model has been trained on all available training objects. However, two different training cases have been chosen. Case A trained the algorithm with all objects, while several objects were assigned the same class. In particular, all chairs, tables and computers have been assigned their corresponding classes 0, 1 and 2. Additionally, this task also evaluated whether the algorithm is capable of generalizing with increasing number of training models for each class. Case B therefore trained the algorithm with all objects again, while this time each object was assigned its individual class. Classes were chosen to range from 0 to 8 for the total of 9 training objects. This case allows to verify whether the detection quality varies according to the training strategy.



**Figure 5.6:** Comparison of training cases in classification results. (A) Each training model has been assigned its corresponding class  $\in \{0, 1, 2\}$  (B) Each training model has been assigned an individual class  $\in \{0, \dots, 8\}$ . Please note that the scale for (table3) has been adapted in order to fully visualize the extent of the data.

Following the experimental results from Section 5.2.2, the test cases themselves were performed on 9 different parameter permutations. Evaluation of the effects of the codebook creation parameters suggested that using only a small amount of clustering and an activation strategy with a low number of activated codewords produced the best results. The cluster factor was therefore changed to take the values 1, 0.9 and 0.8, while the number of activated codewords, and thus the activation strategy parameters, was chosen as 1, 2 and 3. Training and detection were then performed for each scene containing one arbitrarily translated and rotated object, with different levels of Gaussian noise applied, for each of the two training cases and the 9 parameter permutations. Evaluation is performed by averaging the detection results of each of the 9 parameter permutations and regarding the computed averages in relationship to the performed test cases.

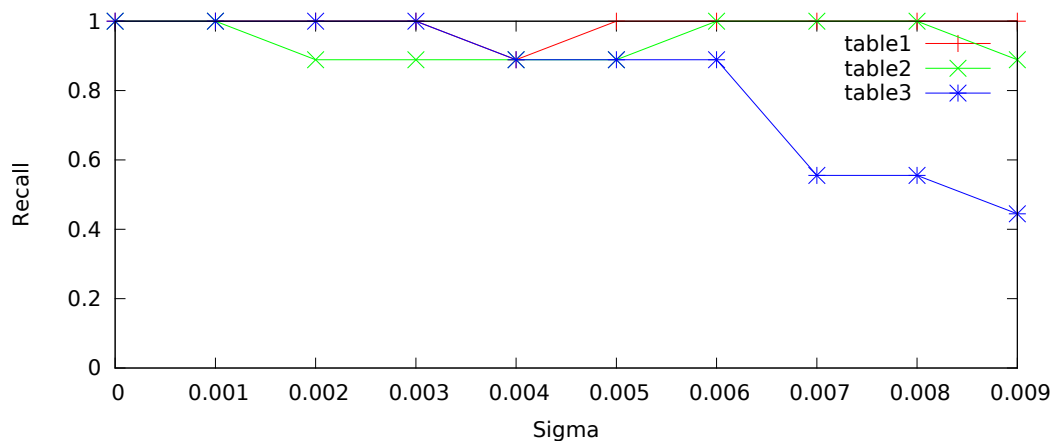
Detection was applied to the corresponding scene containing the object. Figure 5.6 shows the average distance to the ground truth, depending on the chosen value for sigma, computed over all first ranked object hypotheses on the conducted test cases. The different training cases are shown as (A) and (B) in the same graph to allow comparison.

Figure 5.6 shows an excerpt from the resulting classification task and suggests that the detection precision for training case A is lower than for case B in most cases. This indicates that the algorithm loses precision in cases where different models are trained for the same class. An outlier can be seen in 5.6 (table3), which shows that with increasing Gaussian noise applied to the model, the distance to the ground truth rapidly increases for each of the training cases. This can be explained by the characteristics of the training model and will be discussed later on.

In order to check whether this connection is valid, a *two-sample t-test* is performed on the underlying data [Leh86]. Given the average best distances from both training cases, the t-value describes how the samples from both data differ, regarding their weighted variance. It can be regarded as a measure for the significance of the difference between the two sample sets. A value  $t < 0$  indicates that the average of samples from the first dataset is smaller compared to the second dataset. The t-test was performed using training case A as the first dataset and case B as the second one. The results yield a value of  $t = 2.827$ , which indicates that the mean of differences from training case A is higher compared to training case B. In order to determine whether this relationship can be considered coincidental, a probability of 95 % is chosen. Matching these results with the *t-distribution*, a probability of 0.95 requires a t-value of at least 1.653. Since the computed value is higher, the relationship between the two training cases is most likely not caused by coincidence. Furthermore, computing the corresponding confidence given by the t-distribution for the determined t-value yields a probability of 99.74 % that the average distance of training case A is higher than training case B. Following these results, it is valid to conclude that the algorithm does not generalize better with increasing number of training models per class and it is preferable to use distinct classes for each training model.

On the classification task and a total of 10 different values of Gaussian noise, with corresponding sigma ranging up to a value of 0.009, the evaluation achieved an average recall of 0.972 for the best ranked object hypothesis, considering training case A. This means that out of all test cases for training case A, 97.2 % of all object hypotheses with index 0 had a distance to the ground truth of  $< 0.1 m$ . Considering training case B, in which every training object was given its individual class, recall achieved a value of 0.99.

Classification fails especially on table objects, as shown in Figure 5.7. It can be seen that with increasing Gaussian noise on the object, recall drops from 1 for each object to 0.44 for table3. This correlates with Figure 5.6, which shows that for training case A, the average distance to the ground truth increases up to a value of  $0.104 m$ , while the average distance for training case B is still lower at  $0.0791 m$ .



**Figure 5.7:** Average recall on table objects.

The reason for the drop in the detection quality lies in the structure of the tables themselves. With increasing Gaussian noise added to the objects, the small structures that define the table area and legs get scattered around and it therefore gets difficult to extract salient information from the local neighborhood of computed keypoints.

## 5.4 Object Recognition

The task of object recognition was performed on the scenes presented in Section 5.1.1 and Section 5.1.2. The results will be discussed in the following section.

### 5.4.1 Kinect Dataset

Evaluation on the Kinect dataset was again performed on two different training scenarios. Training case A trained the Implicit Shape Model with only the objects that are supposed to appear in the scene, while training case B performed training on the complete number of available training objects, whereas each training object was assigned the corresponding class. Additionally, 9 different parameter permutations were used in the process, identical to the previous test cases. By performing detection on each of the two training cases, it is possible to evaluate how and if the algorithm is able to generalize with increasing number of training models and classes, and how the detection results differ.

For each scene and parameter combination, the detection component creates a list of object hypotheses that is compared to the ground truth data. For each model in the ground truth, the object hypothesis with the minimum distance to

the ground truth is collected and the Euclidean distance is used as a measure for the detection precision. The predicted significance of the detection is represented by the hypothesis index in the detection list and the computed total weight for the chosen hypothesis. The crucial point in the algorithm is the total weight, computed for each of the detected object hypotheses. Since the total number of expected objects is typically unknown, along with the number of expected classes, computing a comparable weight in a most generic way proved difficult. This is supported by the results of the presented evaluation. Each of the scenes contained either a single object or multiple objects, with corresponding ground truth. Each scene was evaluated with a total of 18 different test cases, composed of 9 parameter permutations and 2 training scenarios. The results for training case A are depicted in Table 5.8, which shows the minimum distance from any object in the detection list to the ground truth.

**Table 5.8:** Average results on the minimum distance object hypothesis for training case A. The values for the average distances are given in meters.

Category	Average distance	Average Index	Average Weight
1 (Single Objects)	0.033	7.26	46.34
2 (Multiple Objects)	0.032	6.9	8.71
3 (Occlusions)	0.039	6.11	10.81

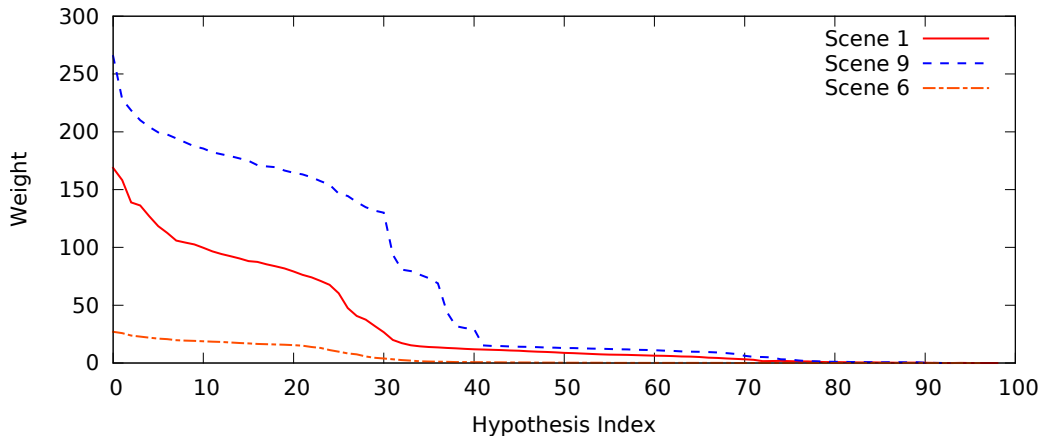
This data shows that while the algorithm is able to detect object hypotheses at the ground truth locations within the specified margin, the computed weights are not comparable and are computed within a wide range.

While Table 5.8 shows the results on the minimum ground truth distance, the significance of an object hypothesis can be determined by the hypothesis index in the detection list. Since the detection list has been sorted by the total weight, an hypothesis with a high rank is supposed to be detected at a higher probability than an object hypothesis with a lower detection rank. Table 5.9 therefore shows the average results on the first ranked hypotheses for the current test cases for training case A.

**Table 5.9:** Average results on the first-ranked object hypothesis for training case A. The values for the average distances are given in meters.

Category	Average distance	Average Weight
1 (Single Objects)	0.538	70.19
2 (Multiple Objects)	0.358	13.66
3 (Occlusions)	0.608	16.87

The average distance of the first ranked hypothesis is above the specified margin at all times. Although a maximum has been detected at the expected location, as



**Figure 5.8:** Average weights for training case A

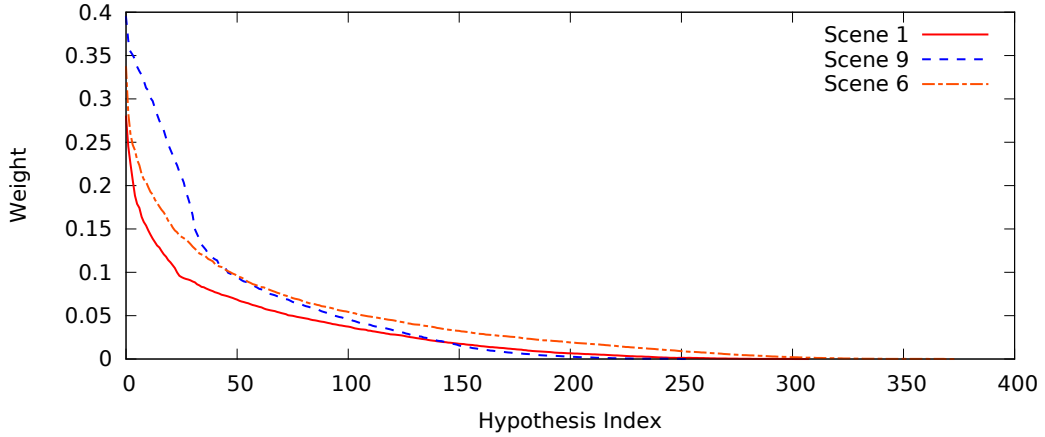
shown by Table 5.8, filtering the results for true positives will not be successful in these cases.

Figure 5.8 also shows how the weight changes with increasing hypothesis rank for selected scenes. The illustrated weight has been computed as the average over all object hypotheses computed on the scene, depending on the current index. Since there are three different categories, each scene has been chosen as a representative of one of the categories. Scene 1 belongs to category 1, containing only one instance of chair1. Scene 9 contains two objects, table1 and chair4, therefore belonging to category 2. Scene 6 represents an occlusion of one object by another, thus containing two different objects and belonging to category 3. For an optimal weighting strategy, the average weights would be computed such that the most likely object hypothesis would be assigned a weight close to 1, while all other unlikely object hypotheses would have been assigned weights close to 0. Such a weighting strategy would allow to compare object hypotheses for their probability and filter out false positives from true positives. However, the graph shows that the weights are not comparable.

**Table 5.10:** Average results on the minimum distance object hypothesis for training case B.

Category	Average distance	Average Index	Average Weight
1 (Single Objects)	0.044	15.25	0.09
2 (Multiple Objects)	0.032	9.639	0.09
3 (Occlusions)	0.046	19.33	0.06

The same applies to training case B, which trained the Implicit Shape Model with all available training objects. This training case is considered to be the



**Figure 5.9:** Average weights for training case B

significant one, since it uses all available information to detect any unknown object instance that matches one of the training models. Accordingly, Table 5.10 supports the results given by Table 5.8. The expected ground truth object has been detected successfully at some index in the detection list. Considering the weight, however, the previous assumptions are also supported.

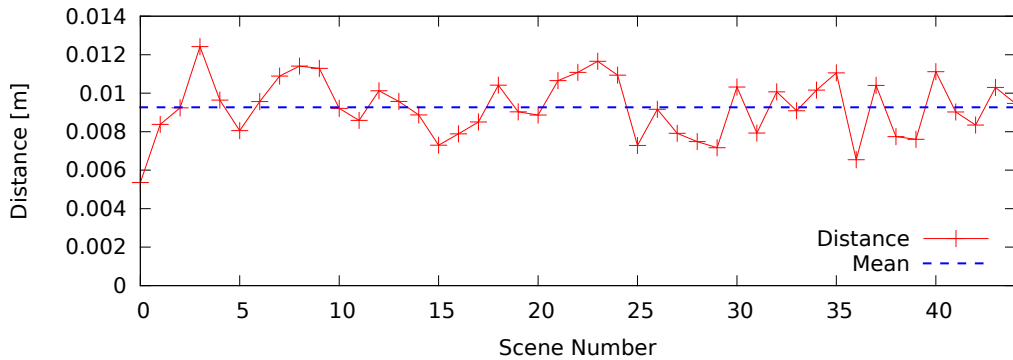
**Table 5.11:** Average results on the first-ranked object hypothesis for training case B.

Category	Average distance	Average Weight
1 (Single Objects)	0.43	0.11
2 (Multiple Objects)	0.306	0.11
3 (Occlusions)	0.261	0.08

The unstable weighting strategy is therefore supposed to be responsible for the results shown in Table 5.11, which shows no improvement over previous results from the restricted training case A. The average distance from the first ranked object hypothesis to the ground truth does not match the chosen margin.

Additionally, Figure 5.9 shows how the weights behave in training case B, where each object has been assigned its individual class and no two objects have the same class. The weighting in this case shows a different weight distribution with comparable weights for each of the scenes. The weight drops significantly for subsequent object hypotheses as expected. However, previous results already indicated that the first ranked object hypotheses do not match the corresponding ground truth data.



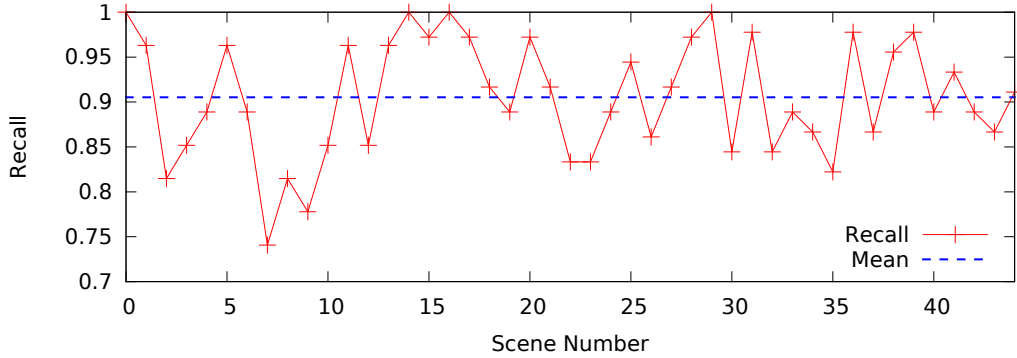


**Figure 5.10:** Average distance of true positives to the ground truth for the Stanford dataset scenes.

### 5.4.2 Stanford Dataset

Evaluation was also performed on the 45 scenes described in Section 5.1.2, which each contain 3 - 5 different training models from the Stanford 3D Scanning Repository with arbitrary rotation and translation. The exact positions and classes are unknown and are to be detected by the Implicit Shape Model. Training has been performed on all training models, while each model has been assigned its individual class. For each scene, ground truth data is available as the expected 3D position, the size of the bounding box and the corresponding rotation quaternion.

The Stanford dataset consists of 45 different scenes. The first 15 scenes contain each 3 objects, randomly chosen from the total of 6 different objects with arbitrary rotation and translation, 15 scenes each containing 4 objects, and the last 15 scenes each containing 5 different objects. For each of the 45 scenes, the parameters for the clustering and the activation strategy have again been changed to create 9 different test cases. Figure 5.10 shows the average distances for each scene, computed over all true positives on each of the 9 different parameter sets. Please note that the different objects from the Stanford dataset are typically small objects. The scale, however, is the same and is given in meters. Due to the nature of the scanning process conducted on the real objects from the Stanford dataset, the point cloud density is higher compared to point cloud data coming from the Kinect sensor. For this dataset, the maximum allowed distance from a object detection to the corresponding ground truth was set to  $0.02\text{ m}$ . In order to apply the detection process to smaller objects, further parameters of the algorithm had to be adapted to the changed object dimensions. Keypoints are still extracted using the voxel grid approach, while the voxel grid size has been changed to  $0.02\text{ m}$ . Similarly, the local point neighborhoods from the descriptor computation were also adapted to a smaller size. The bandwidth of the Mean-Shift algorithm has been changed to



**Figure 5.11:** Recall on the Stanford dataset scenes. Please note that the scale of the y-axis has been adapted.

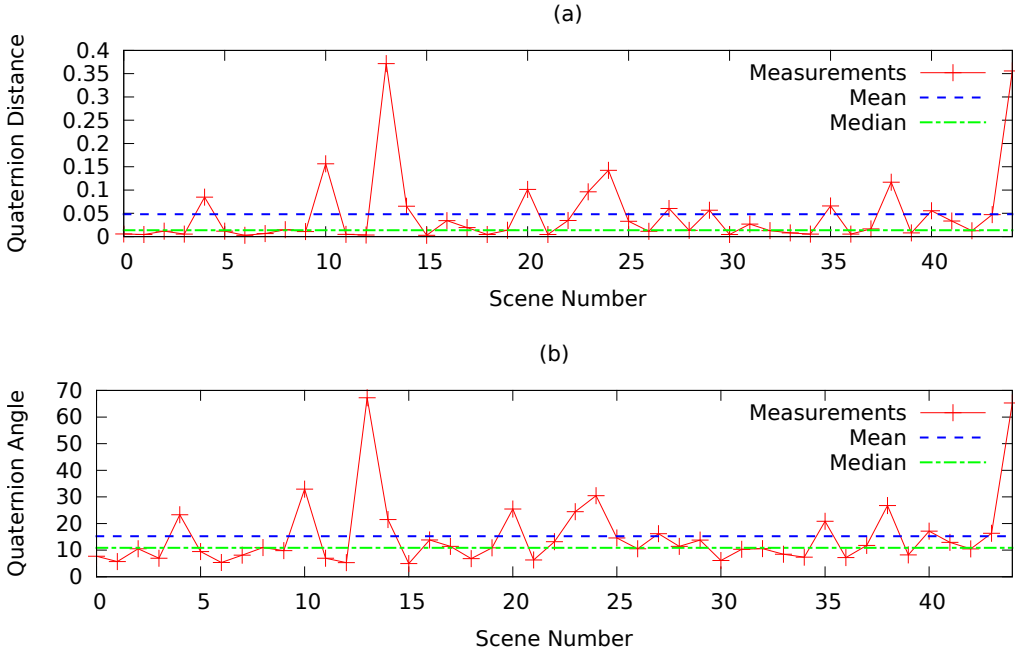
$0.05 m$ , which allows a reasonable number of detected object hypotheses and can still detect objects located close to one another.

Figure 5.10 shows that the average distance of all best ranked object hypotheses is below the chosen threshold at all times, with a mean distance of  $0.0093 m$  and a median of  $0.0092 m$ . With the mean distance and the median approximately the same value and the standard deviation of  $0.0015$ , these results indicate that the detection component is able to produce stable object hypotheses for the presented cases.

The recall on each test scene is shown in Figure 5.11. Over all parameter combinations and ground truth objects, the first ranked object hypothesis for each ground truth object is assumed a true positive if the distance is below the chosen threshold of  $0.02 m$ . The graph therefore shows that the true object location is correctly identified at an average of 90.52 %, with a lowest recognition rate of 74.07 %. This test case was also used to evaluate the extraction of the object’s bounding box. While the bounding box comprises an orientation and a size, the precision for extracting the bounding box size is measured by the Euclidean distance from the size vector to the ground truth size. For the given test cases, the size was extracted by an average variance of  $3.67 \times 10^{-7} m$ , which indicates good results. Measuring the quality of the orientation extraction is done in two ways. Since orientations are represented by quaternions, the minimum angle between two unit quaternions  $\mathbf{q}_1$  and  $\mathbf{q}_2$  can be computed by:

$$\theta = \cos^{-1}(2\langle \mathbf{q}_1, \mathbf{q}_2 \rangle^2 - 1) \quad (5.1)$$

Since quaternions can be represented by a rotation axis and a corresponding rotation angle around this axis,  $\theta$  is retrieved by projecting the axis of one quaternion to the second quaternion axis and computing the angle that is required for



**Figure 5.12:** Average orientation distances over all detected scenes. (a) Each data point represents the average quaternion distance to the ground truth, computed over all true positives. (b) Each data points represents the average quaternion angle over all true positives, compared to the ground truth.

one of the quaternions to achieve identical rotation. Additionally, a distance measure  $\in [0, 1]$  between unit quaternions can be computed, which is 0 in cases where the orientations are equal and 1 when the orientations are opposite:

$$d(\mathbf{q}_1, \mathbf{q}_2) = 1 - \langle \mathbf{q}_1, \mathbf{q}_2 \rangle^2 \quad (5.2)$$

Figures 5.12 (a) and (b) show the correlation between quaternion distance and angle. Computed over all scenes and true positive detections, the mean distance of extracted orientations to the ground truth data is 0.0482 with a standard deviation of 0.0781, and a median of 0.0137. Given that a distance of 0 represents optimal matching orientations, the average precision for the extraction of object orientations in unknown scenes can therefore be given by  $1 - 0.0482 = 0.9518 \triangleq 95.18\%$ . It can be seen that (a) and (b) correlate, while (b) demonstrates a mean angle difference of  $15.25^\circ$  with a standard deviation of  $12.98^\circ$  and a corresponding median of  $10.87^\circ$ . Allowing a maximum orientation angle of  $20^\circ$ , the data indicates that 87.16% of true positives have been assigned an orientation that matches the expected ground truth.

**Table 5.12:** Results from the Stanford dataset. Shown are best hypotheses for the current scene and the votes that contributed to the detection. The individual classes are represented by the color of their votes. The bounding box has been computed using the contributing votes.

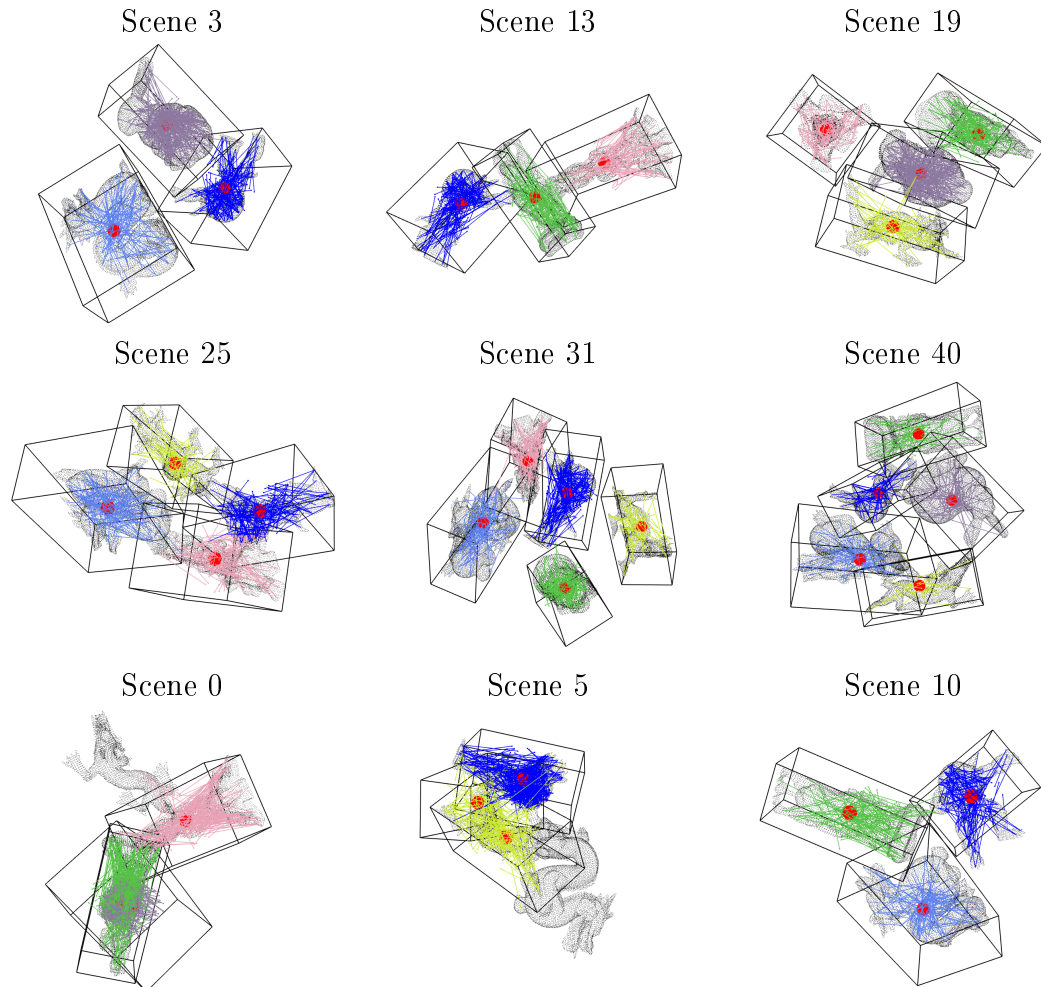


Table 5.12 shows some results from the detection component on the scenes from the Stanford dataset. The first two rows show correct object detections, the bottom row additionally reveals some misdetections. In case of Scene 0, two objects have been identified correctly, although one hypothesis has been detected at an unexpected location nearby an already detected object. This is probably caused by mismatches between features and codewords and can be avoided by an adapted weighting strategy. In case of Scene 5, two objects have been identified correctly, while an object hypothesis of the same class was detected at a nearby position. This is most likely caused by an undervalued bandwidth parameter.

Scene 10 shows three correct object detections, while the extracted bounding box for the upper right object (Armadillo) is misaligned.

## 5.5 Performance

Estimating the performance requirements of the presented method depends on several criteria. First of all, the algorithm computes features on either training objects or the input point cloud used for detection. Thus, runtime varies depending on the chosen keypoint detector and descriptor. Furthermore, since keypoints and descriptors usually need to perform nearest neighbor searches on the point cloud, point size and density also have an influence. The training procedure usually detects features on a number of different training models. The point cloud extent is limited to the dimensions of the current training objects. When performing the detection procedure on an input point cloud captured with a depth camera, the point cloud additionally contains the surrounding area of the objects of interest. However, the point cloud density in the training case is typically higher, since the training objects have been captured from several viewpoints in order to reproduce as much detail as possible.

The activation procedure matches each feature with a number of codewords, according to the chosen activation strategy. Runtime depends equally on the number of features and codewords, as well as on the number of activated codewords per feature. During detection, runtime depends on the number of generated votes and the number of classes. Each class is assigned its individual voting space, in which the corresponding votes for the given class will be cast. Mean-Shift Mode Estimation then searches for maximum density regions in each voting space, which also requires searching for nearest votes within the specified kernel bandwidth.

**Table 5.13:** System specifications for the evaluation system.

Operating System:	Kubuntu 13.04 with GNU/Linux 3.8.0
Processor:	Intel Core i7 920 @ 2.67 GHz
Cores:	8
Architecture:	64 Bit x86-64
Working Storage:	6 GB DDR3 @ 1066 MHz

The specifications for the system used for evaluation are shown in Table 5.13. On this system, the runtime has been evaluated on the test cases conducted on the Kinect dataset, as described in 5.4.1. The average training time was 8.26 s with a standard deviation of 7.89. The average detection time on the described evaluation system was 14.06 s and a standard deviation of 21.96. Higher runtime for the detection is caused by the increasing number of detected features on the

**Table 5.14:** Summary of parameters.

Keypoint detector:	VoxelGrid
Descriptor:	SHOT
Local Reference Frame:	SHOT-NA-LRF
Clustering:	K-Means using clustering factor
ClusterFactor:	$\in \{1, 0.9, 0.8\}$
Activation strategy:	KNN
Activation strategy K:	$\in \{1, 2, 3\}$

point clouds, as well as the complexity of the Mean-Shift algorithm. Additionally, it is important to note that the described runtime values do not incorporate the time for normal computation and consistent normal orientation. This information was computed beforehand and stored along with the point cloud data, since computing normals for the evaluation test cases can be outsourced in order to save computation time. However, in a practical usage scenario, training can be performed offline prior to detection, while only the detection component would require the robot to pause while searching for object hypotheses. Estimating consistent normal orientation is only needed on the training models. Using Kinect data at detection, however, normals can be computed efficiently using integral image normal estimation [HRD<sup>+</sup>12] [HHRB11], therefore still enabling online detection.

## 5.6 Discussion

During evaluation, a specific set of parameters proved beneficial, as shown in Table 5.14. These parameter combinations showed the best results in the evaluated test cases. However, the results differ among the datasets, which will be discussed in detail in this section.

First of all, computing comparable weights has proven to be difficult in the given context. The training objects that have been initially used for training are supposed to represent the complete objects, independently from viewpoints. Features are computed on the front and back sides equally. During detection, the object is naturally visible only from the side that is oriented toward the viewpoint. Activation can therefore only activate codewords that have been created from features that correspond with the visible region. The desired weighting should be chosen such that a probability of 1 equals the highest probability, indicating that the object has been found. However, the training model consists of an object representation that features as much detail as possible captured from different viewpoints in order to allow detection from different viewpoints. During detection, however, object instances are captured with a camera, therefore matching the

training data only from the specific camera viewpoint. Detection will therefore most likely not activate every single codeword that has been originally trained on opposite or occluded positions on the object. As a consequence, the maximum probability will never be reached.

Additionally, the right choice of keypoints is crucial. The algorithm requires the voting space to contain high-density regions in order to robustly detect maxima at the corresponding locations. A high number of votes correlates with a high number of keypoints, along with the choice of the clustering and activation strategy. Given the point cloud resolution from the Kinect data only, it is difficult for existing keypoint detectors to extract enough salient points on the surface by solely relying on point cloud data. The depth map and distance resolution of the Kinect camera are not high enough to robustly allow keypoint detectors to find enough distinguishable keypoints on each model that the presented Implicit Shape Model is able to work upon. That is why keypoint extraction using voxel grid sampling showed better results compared to traditional keypoint detectors in 5.2.1. However, this approach also brings problems in the voting space. Since a multitude of features are created uniformly on the point cloud, the number of votes rises with the number of activated codewords per feature. Since the object positions are unknown, the Mean-Shift Mode Estimation searches for maxima in every location of the point cloud. This causes the algorithm to detect a high number of object hypotheses, along with the corresponding total weights. However, since weighting does not provide comparable results, false positives cannot be filtered for their probability.

The original Implicit Shape Model approach was designed to allow multiple training objects for each class. The idea behind this concept is, that while full rotation invariance is difficult to achieve using 2D images, the object is trained for the same class using images captured from different angles. The results of this thesis, however, showed that this approach is not well applicable to 3D objects. Training different objects for the same class does not cause the Implicit Shape Model to generalize but rather causes the detection quality to drop. However, this can be revised partly by using separate classes for each training object.





# Chapter 6

## Conclusion

This thesis investigated existing approaches to object detection using Implicit Shape Models aiming to develop an algorithm for 3D object recognition. In contrast to existing approaches, the presented algorithm is highly customizable and can be executed with different features. Furthermore, it has been investigated how the training information can be used to extract the the bounding box during detection. The detection was performed using a hough-based algorithm, however, detecting objects at continuous locations. During the evaluation, the use of uniformly sampled keypoints in combination with the SHOT descriptor proved beneficial compared to other combinations of descriptors and keypoint detectors. The evaluation also showed that the basic principle of Implicit Shape Model approaches, i.e. the creation of a codebook of geometric appearances, cannot easily be adapted to 3D object recognition and does not show major advantages compared to using a codebook of full size. Analyzing combinations of codebook size and activated codewords at the same time revealed that the process of 3D codebook creation on local geometric features rather reverses the detection capability. However, the algorithm proved its functionality and showed accurate results in the task of object classification. It has been proven that the bounding box and therefore the object orientation can be extracted adequately in such cases. For use in cluttered and noisy scenes captured from a single viewpoint, however, the 3D adaptation of Implicit Shape Models in combination with a global codebook did not prove beneficial. In this context, approaches without the process of clustering features into codewords might already prove to be sufficient. The main benefit of Implicit Shape Models as compared to other detection approaches lies in the process of codebook creation and the activation strategy, which both control how votes are created in the voting space. In the evaluation, different test cases have been conducted that illustrated the usage scenarios in which the method is able to correctly identify object instances. In addition, the limitations of the proposed algorithm have also been demonstrated.

## 6.1 Further Prospects

In combination with detecting objects in scenes captured with a depth camera, i.e. scenes that contain additional objects besides the objects of interest, different weighting strategies can be evaluated. Problems with the current weighting strategy occur when training multiple training objects for the same class. The weights have to be adapted such that strong object detections naturally are computed with a higher probability from the individual vote weights, compared to less significant object detections. The total weight is supposed to represent a true probability, in order to correctly filter between significant and insignificant object hypotheses.

Regarding the number of keypoints, the detection algorithm highly depends on accurate and numerous features. Using the SHOT descriptor proved to be a good tradeoff in terms of performance and precision. In the presented task, it was not acceptable for the object recognition process to require computation times up to several hours. Since the voting can only work accurately with a high number of votes, a high number of features is therefore inevitable. There are, however, different descriptor types that can work with additional information as well. Notably, there exist several descriptors that incorporate color information as well, like an RGB adaption to the PFH descriptor or the SIFT descriptor [SAS07]. Using 3D information captured from a Kinect camera alone poses problems, since it is difficult to extract enough significant descriptions from objects that do not contain sufficient discriminative information themselves. This can either be improved by increasing the point cloud resolution with a different type of sensor, or by incorporating additionally available information like color.

When all votes have been cast into the voting space, adapting the Mean-Shift Mode Estimation can further improve the detection precision. The used Mean-Shift algorithm requires the knowledge of the bandwidth parameter that controls how the individual samples are combined within the kernel in order to form the mean shift vector. The bandwidth can be interpreted as the minimum distance between detected maxima. Assuming a fixed bandwidth, however, is not beneficial at all times. For a region in the voting space with a high vote density, a large bandwidth causes the kernel gradient estimation to compute the mean shift vector over a large number of data samples. In cases in which the local vote density varies inside the kernel bandwidth, the Mean-Shift algorithm will reach a local maximum and stop shifting, although the true maximum has not yet fully been reached. A small bandwidth parameter on the other hand allows the algorithm to improve precision, since it is now able to refine maxima in cases where a large bandwidth parameter would have caused the algorithm to stop. A small bandwidth, however, also results in an increased number of detected maxima. Therefore, it is also possible that different maxima are detected at nearby locations, which would otherwise have been merged and probably result from the same object hypothesis. An

adaptation to the Mean-Shift algorithm has been proposed in [CRM11]. Starting from the fixed bandwidth algorithm, the authors introduce the *Variable Bandwidth Mean Shift* algorithm, that assigns each data point an individual bandwidth parameter, based on the point density at the data point location. The individual point-based bandwidth parameters then cause the algorithm to assign each data point an individually scaled kernel function that is adapted to each point's neighborhood and lead to a small kernel in regions with high point density, while the kernel is larger in regions with a low point density. Incorporating this algorithm is suited to improve the detection precision.

The evaluation also showed that the process of codebook creation does not have positive influence on the detection. Rather, detection precision decreases when more features are clustered into codewords. Using a codebook that has been clustered from different features was initially motivated by the large amount of detected features and the necessity to train multiple views of the same object in order to achieve a type of rotation invariance. This necessity is not given when using 3D information, since rotation invariance can be achieved implicitly. Compared to the ISM approach for 2D images, the number of features and the performance of state-of-the-art computers does not legitimate the benefits of a reduced codebook size compared to the increased computation time. However, further work can be done by evaluating the relationship between using a local codebook and a global codebook. While using a global codebook seemed reasonable in the context of this thesis, using a local codebook that only clusters features from training models within the same class might reduce the negative effects of the global clustering. Additionally, performing clustering on a per-model basis is potentially suited to increase precision.

Detecting the object position alone provides a good starting point for further examinations. This thesis also discussed how the bounding box for a detected object hypothesis can be extracted from the information stored within the votes that contributed to the specific hypothesis. However, along with the bounding box, the segmentation of the object within the scene is of interest. Segmentation can be achieved by applying a region growing algorithm on the point cloud data starting from the computed bounding box position and evolving the segmentation until the bounding box no longer fully includes the segmented object. Applying a pre-segmentation to the point cloud using supervoxel clustering [PASW13] creates a connected graph of small regions with similar size and normal distributions, which can further be integrated into the region growing. The growing can then stop when a considered supervoxel cluster is outside the bounding box. To improve robustness, the bounding box considered for the region growing can be scaled by a certain factor, such that small variations in size and orientation of the object can be compensated.



# Appendix A

## Appendix

This section gives additional information about implementation details. Each component is described with its required parameters and dependencies. Additionally, the developed user interface for visualizing training and detection is presented.

### A.1 Framework Overview

The developed algorithm for Implicit Shape Model based object recognition has been encapsulated in a generic framework that allows for fast and easy implementation of new algorithms, as well as means to serialize and deserialize data and parameters. In order to be able to easily incorporate new algorithms and see their effects on the detection capabilities, the framework uses a templated factory pattern. Each individual basic component, such as feature detection or clustering, provides its generic interface for communication with the main controlling instance. The precise specializations implement the interface, but their behaviour differs accordingly. Additionally, the system needs to be able to serialize training data into files, which can be read on later purpose by the detection component.

In this context, the Javascript Object Notation (JSON) was used for serializing and deserializing data and parameters, since JSON can be used to provide human-readable configuration files. In order to separate between parameters and training data, serializing the Implicit Shape Model creates two distinct files. The configuration file contains all parameters, as well as information about the precise type of specializations of the individual components. It also includes a relative path to the data file that contains the corresponding training data. Separating between parameters and training data was a necessary step to allow an easy configuration.

## A.2 Parameters

The precise parameter configuration for the training and detection procedures are determined by a configuration file using the JSON standard. A typical configuration file is given below.

---

```

1 {
2   "ObjectConfig" : {
3     "Children" : {
4       "Clustering" : {
5         "Parameters" : {
6           "CbIndex" : 0.5,
7           "CentersInit" : "FLANN_CENTERS_KMEANSPP",
8           "ClusterFactor" : 0.2,
9           "Iterations" : 1000
10        },
11       "Type" : "KMeansFactor"
12     },
13     "Codebook" : {
14       "Children" : {
15         "ActivationStrategy" : {
16           "Parameters" : {
17             "K" : 2
18           },
19         "Type" : "KNN"
20       }
21     },
22     "Parameters" : {
23       "DetectionBestStrategy" : true,
24       "UseClassWeight" : true,
25       "UseMatchingWeight" : true,
26       "UseVoteWeight" : true
27     }
28   },
29   "Features" : {
30     "Parameters" : {
31       "Radius" : 0.1,
32       "ReferenceFrameRadius" : 0.2,
33       "ReferenceFrameType" : "SHOTNA"
34     },
35     "Type" : "SHOT"

```

```

36     },
37     "Keypoints" : {
38         "Parameters" : {
39             "LeafSize" : 0.1
40         },
41         "Type" : "VoxelGrid"
42     },
43     "Voting" : {
44         "Parameters" : {
45             "Bandwidth" : 0.2,
46             "BestK" : -1,
47             "Kernel" : "Gaussian",
48             "MaxIter" : 1000,
49             "MinThreshold" : 0.0,
50             "Threshold" : 0.001
51         },
52         "Type" : "MeanShift"
53     }
54 },
55 "Parameters" : {
56     "BoundingBoxType" : "MVBB",
57     "ConsistentNormals" : true,
58     "ConsistentNormalsK" : 10,
59     "NormalRadius" : 0.05,
60     "NumThreads" : 0,
61     "UseVoxelFiltering" : false,
62     "VoxelLeafSize" : 0.01
63 }
64 },
65 "ObjectData" : "data.ismd"
66 }

```

The parameter "ObjectData" specifies the path to the file containing training data. When this parameter is missing, an empty Implicit Shape Model is assumed, however still applying the specified parameters. The empty model can then be used for training.

Each individual component is described with its associated parameters and the type of the specialization is given by the type parameter. The following section describes the available parameters and components. It is important to note that the framework consists of multiple modules, implemented using polymorphism.

Parameters from base classes are therefore inherited to the derived class. The following list of modules describes only parameters for the current class.

### A.2.1 General

The main entry point for Implicit Shape Model based object detection is the `ImplicitShapeModel` class, which provides the interface for providing training classes and detecting the learned model in unclassified point clouds.

<b>Class</b>	<code>ImplicitShapeModel</code>
<b>Description</b>	The main class for Implicit Shape Model based object recognition. It performs training as well as detection and can be serialized and deserialized using JSON files.
<b>Parameter</b>	<b>Description</b>
<code>BoundingBoxType</code>	The type of bounding box that is used in the training process. The precise type also determines the type of the extracted bounding box for detected objects. Allowed values are "MVBB" for the minimum volume bounding box and "AABB" for the axis aligned bounding box. The default value is "MVBB".
<code>NumThreads</code>	Determines the number of threads to use. A value of 0 automatically detects the best number of threads. The default value is 0.
<code>NormalRadius</code>	The radius for the nearest neighbors computation used in the process of extracting normal. The default value is 0.05.
<code>UseVoxelFiltering</code>	Can be used to preprocess point clouds and creating equal point densities. Can be disabled for most cases, since keypoints and descriptors automatically cope with point density variations. The default value is false.
<code>VoxelLeafSize</code>	The size of the voxel grid in case voxel filtering has been enabled. The default value is 0.01.
<code>ConsistentNormals</code>	In case normals are not available for the current training object, determines whether a consistent normal orientation should be propagated on the object surface. The default value is true.
<code>ConsistentNormalsK</code>	The number of nearest neighbors from which the Riemannian Graph is constructed. The default value is 10.



## A.2.2 Keypoints

The Keypoints module computes keypoints on the input point cloud, depending on the chosen specialization.

- Keypoints

<b>Description</b>	Base class for keypoint detection. The input is a cloud with associated normals.
--------------------	--

- KeypointsHarris3D

<b>Description</b>	Detects keypoints using the PCL adaption of the Harris corner detector.
<b>Baseclass</b>	Keypoints
<b>Type</b>	"Harris3D"
Parameter	Description
Radius	The radius for nearest maxima search. The default value is 0.05.
NonMaxSupression	Set to true to perform non-maximum suppression. The default is true.
Threshold	Threshold for corner detection. Only valid if non-maximum suppression is activated. The default value is 0.0001.
Refine	Set to true to refine detected corners. The default value is true.
ResponseMethod	The response method. The default value is "HARRIS". Alternatives are "NOBLE", "LOWE", "TOMASI" and "CURVATURE".

- KeypointsISS

<b>Description</b>	Detects keypoints using Intrinsic Shape Signatures.
<b>Baseclass</b>	Keypoints
<b>Type</b>	"ISS3D"
<b>Parameter</b>	<b>Description</b>
SalientRadius	The radius of nearest neighbor search for computation of the covariance matrix. The default value is 0.1.
Gamma21	Specifies an upper boundary for eigenvalue decomposition on the covariance matrix on the first and second eigenvalue. The default value is 0.975.
Gamma32	Specifies an upper boundary for eigenvalue decomposition on the covariance matrix on the second and third eigenvalue. The default value is 0.975.
NonMaxRadius	The radius for the application of non-maximum suppression. The default value is 0.05.
MinNeighbors	The minimum number of nearest neighbors when performing non-maximum suppression. The default value is 5.

- KeypointsVoxelGrid

<b>Description</b>	Detects keypoints using a voxel grid superimposed on the point cloud.
<b>Baseclass</b>	Keypoints
<b>Type</b>	"VoxelGrid"
<b>Parameter</b>	<b>Description</b>
LeafSize	The leaf size for the voxel grid. This parameter has direct influence on the keypoint density.

### A.2.3 Features

The features component uses computed keypoint positions to extract descriptors and local reference frames on the input point cloud. The output is a feature point cloud containing all information on keypoint positions that are used in the process of training and detection.

- Features

<b>Description</b>	The base class for feature detection.
<b>Parameter</b>	<b>Description</b>
ReferenceFrameRadius	The radius for nearest neighbor search when detecting local reference frames. The default value is 0.2.
ReferenceFrameType	The type of the local reference frame to use. The default value is "SHOTNA". Alternatively, "SHOT" can be used.

- FeaturesPFH

<b>Description</b>	Computes features using Point Feature Histograms.
<b>Baseclass</b>	Features
<b>Type</b>	"PFH"
<b>Parameter</b>	<b>Description</b>
Radius	The radius for nearest neighbor search. The default value is 0.1.

- FeaturesFPFH

<b>Description</b>	Computes features using Fast Point Feature Histograms.
<b>Baseclass</b>	Keypoints
<b>Type</b>	"FPFH"
<b>Parameter</b>	<b>Description</b>
Radius	The radius for nearest neighbor search. The default value is 0.1.

- FeaturesSHOT

<b>Description</b>	Computes features using Signature of Histograms of Orientations.
<b>Baseclass</b>	Features
<b>Type</b>	"SHOT"
<b>Parameter</b>	<b>Description</b>
Radius	The radius for nearest neighbor search. The default value is 0.1.

### A.2.4 Clustering

The clustering component determines how the features detected on the training objects are clustered into codewords. Different types of clustering strategies have been implemented.

- Clustering

<b>Description</b>	The base class for any clustering algorithm. The input is a list of detected features.
--------------------	--

- ClusteringNone

<b>Description</b>	Does not perform any clustering. The input features are each associated with their individual cluster.
--------------------	--

<b>Baseclass</b>	ClusteringKMeans
------------------	------------------

<b>Type</b>	"None"
-------------	--------

- ClusteringAgglomerative

<b>Description</b>	Performs agglomerative clustering.
--------------------	------------------------------------

<b>Baseclass</b>	ClusteringKMeans
------------------	------------------

<b>Type</b>	"Agglomerative"
-------------	-----------------

Parameter	Description
Threshold	The threshold that determines when the clustering should stop. The threshold depends on the chosen descriptor type and is given in the descriptor space. The default value is 1.2.

- ClusteringKMeans

<b>Description</b>	The base class for K-Means based clustering algorithms.
--------------------	---

<b>Baseclass</b>	Clustering
------------------	------------

Parameter	Description
Iterations	The maximum number of algorithm iterations. The default value is 1000.
CentersInit	Determines how the initial cluster centers are created. The default value is "FLANN_CENTERS_KMEANSPP". Alternatives are "FLANN_CENTERS_GONZALES" and "FLANN_CENTERS_RANDOM".
CbIndex	The cluster boundary index. This internal parameter is used when searching the K-Means tree. The default value is 0.5.

- ClusteringKMeansCount

<b>Description</b>	Performs K-Means clustering on a specified number of clusters.
<b>Baseclass</b>	ClusteringKMeans
<b>Type</b>	"KMeansCount"
<b>Parameter</b>	<b>Description</b>
ClusterCount	The number of desired features. In case the number of input features is smaller, this parameter has no effect. The default value is 10.

- ClusteringKMeansFactor

<b>Description</b>	Performs K-Means by multiplying the number of input features by a specified factor.
<b>Baseclass</b>	ClusteringKMeans
<b>Type</b>	"KMeansFactor"
<b>Parameter</b>	<b>Description</b>
ClusterFactor	The cluster factor that is multiplied by the number of input features. The default value is 0.2.

- ClusteringKMeansHartigan

<b>Description</b>	Performs K-Means by determining the cluster count from Hartigan's rule.
<b>Baseclass</b>	ClusteringKMeans
<b>Type</b>	"KMeansHartigan"
<b>Parameter</b>	<b>Description</b>
MaxK	Hartigan's rule requires K-Means to run multiple times from $k = 1$ to the parameter of MaxK. The default value is 10.

- ClusteringKMeansThumbRule

<b>Description</b>	Performs K-Means by using a thumb rule to determine the cluster count. The resulting number of clusters is determined by $\sqrt{\frac{x}{2}}$ , while $x$ is the number of input features.
<b>Baseclass</b>	ClusteringKMeans
<b>Type</b>	"KMeansThumbRule"

### A.2.5 Codebook

The codebook stores the individual codewords and maps each codeword to its list of activation vectors. It is also responsible for performing the activation during training and detection.

<b>Class</b>	Codebook
<b>Description</b>	Contains the individual codewords and the associated activation vectors.
<b>Parameter</b>	<b>Description</b>
UseClassWeight	Set to true to activate statistical weights. The default value is true.
UseVoteWeight	Set to true to activate center weights. The default value is true.
UseMatchingWeight	Set to true to activate matching weights. The default value is true.
DetectionBestStrategy	Experimental parameter. Set to true if the detection stage should only activate using the <i>Best</i> strategy. Set to false to use the same activation strategy as in training. The default value is false.

### A.2.6 Activation Strategy

The ActivationStrategy class is provided with a detected feature and the computed codebook and returns a list of activated codewords according to the chosen specialization.

- ActivationStrategy

<b>Description</b>	The base class for any activation strategy. The input is a detected feature and a codebook.
--------------------	---

- ActivationStrategyBest

<b>Description</b>	Activates the best matching codeword for the current feature.
<b>Baseclass</b>	ActivationStrategy
<b>Type</b>	"Best"

- ActivationStrategyKNN

<b>Description</b>	Activates the $K$ best matching codewords for the current feature.
<b>Baseclass</b>	ActivationStrategy
<b>Type</b>	"KNN"
<b>Parameter</b>	<b>Description</b>
K	The number of best codewords to activate for the current feature. The default value is 2.

- ActivationStrategyThreshold

<b>Description</b>	Activates all codewords whose distance to the current feature is below a threshold.
<b>Baseclass</b>	ActivationStrategy
<b>Type</b>	"Threshold"
<b>Parameter</b>	<b>Description</b>
Threshold	The minimum distance in descriptor space between any activated codeword and the current feature. The default value is 1.0.

### A.2.7 Voting

The voting module is responsible for the weighted voting process and searches for maxima. It also computed the votes that contributed to each maxima and estimates an averaged bounding box. The output from the voting module is forwarded to the user and presents a list of detections with additional information.

- Voting

<b>Description</b>	Base class for voting and maxima search. Maxima search is performed on each class individually and the results are merged. Additionally, the average bounding box on all contributing votes is computed.
<b>Parameter</b>	<b>Description</b>
MinThreshold	The minimum weight for resulting maxima. Maxima with a weight lower than the threshold will be discarded. A value of 0 indicates that all detected maxima should be returned. The default value is 0.
BestK	Indicates whether only the K best maxima should be returned. A value of -1 indicates that the number of maxima is not restricted. The default value is -1.
AverageRotation	Set to true if the average rotation should be computed.

- VotingMeanShift

<b>Description</b>	Detects maximum density regions in the voting space using Mean-Shift Mode Estimation.
<b>Baseclass</b>	Voting
<b>Type</b>	"MeanShift"
<b>Parameter</b>	<b>Description</b>
Bandwidth	The kernel bandwidth parameter specifies the width of the used kernel. The default value is 0.2.
Threshold	The threshold that determines when the algorithm considers a maximum found. The default value is 0.001.
MaxIter	The maximum number of iterations. This value guarantees that the algorithm will always stop. The default value is 1000.
Kernel	The kernel type. The default value is "Gaussian". Alternatively, "Uniform" can be used.



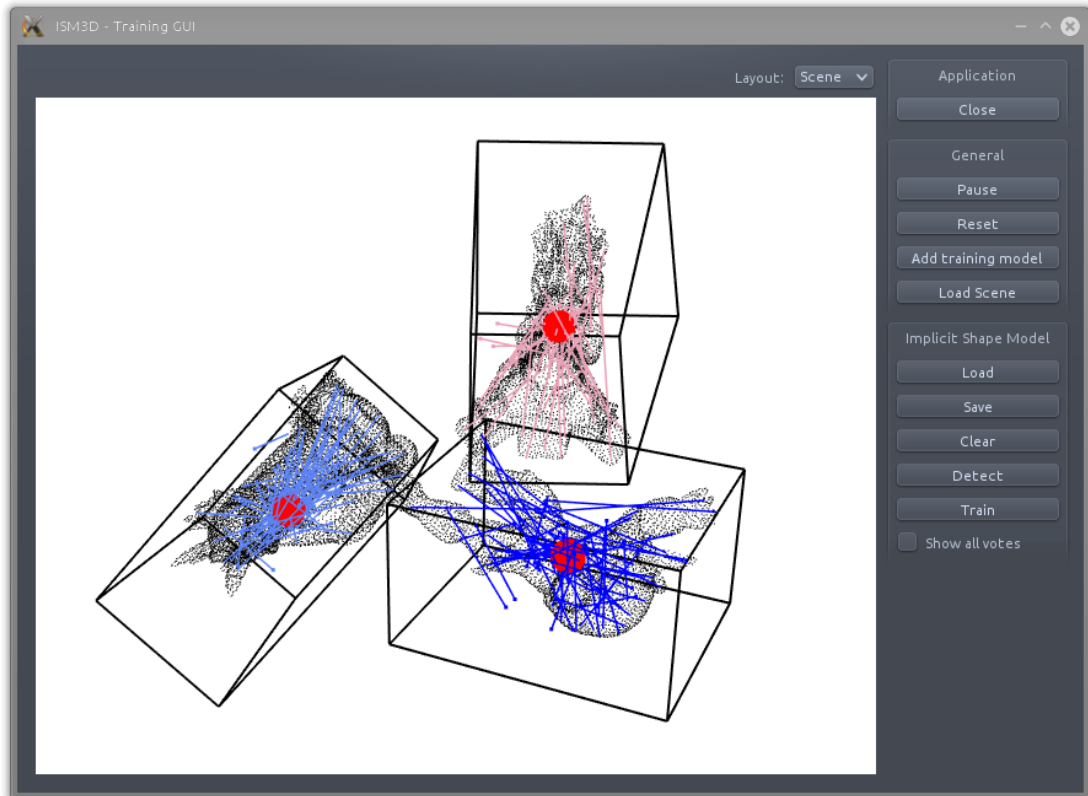
- VotingHough3D

<b>Description</b>	Detects maximum density regions using a discrete hough space.
<b>Baseclass</b>	Voting
<b>Type</b>	"Hough3D"
<b>Parameter</b>	<b>Description</b>
MinCoord	Specifies the minimum extent of the hough space. The default value is "[ -5, -5, -5 ]".
MaxCoord	Specifies the maximum extent of the hough space. The default value is "[ 5, 5, 5 ]".
BinSize	Specifies the size of each hough bin. The default value is "[ 0.2, 0.2, 0.2 ]".
UseInterpolation	Set to true if maxima should be interpolated between neighboring bins.
RelThreshold	A relative parameter $\in [0, 1]$ specifying the relation between the any returned vote and the weight for the highest ranked vote. The default value is 0.8.

### A.3 Training-GUI

In order to be able to test the algorithm and visualize the results, a graphical user interface using the Qt UI framework has been developed. This application allows the user to create a new Implicit Shape Model, add a number of training models and train the algorithm. The trained data can then be saved to two JSON files, of which one contains the configuration and the other the corresponding training data. By importing the trained Implicit Shape Model, the user is able to load an arbitrary scene and apply the detection process to it. The results are visualized in a 3D view. Figure A.1 shows the user interface.

Configuration of the Implicit Shape Model is done by changing the parameters in the corresponding JSON file. At startup, the application first creates a default ISM representation. When no initial configuration file is present, the default ISM can be saved to a file, where the necessary parameters can be altered. Loading the file again applies the changes.



**Figure A.1:** The developed Training-GUI with a loaded scene from the Stanford dataset and the visualized detection process.

## A.4 External Components

The implementation of this thesis uses a variety of different libraries that helped in the implementation process. These are:

- PCL 1.7 - Point Cloud Library [RC11] [Poi14]  
The Point Cloud Library presents a variety of algorithms and methods to analyze 3D point clouds. Individual points can contain multiple information like normal and curvature.
- Boost 1.49 [Boo14]  
The Boost library is a general purpose utility library that covers a multitude of different use cases, from graph analysis to threading and synchronization.
- Eigen 3.1.2 [Eig14]  
Eigen is a mathematical library providing common data structures and algorithms to easily work with matrices and vectors. Eigen has been used

for eigenvalue decomposition, as well as handling transformations and quaternion mathematics.

- Qt 4.8 [QtP14]  
Along with the object detection algorithm, which has been implemented as a library, different user interfaces have been developed with the Qt UI framework in order to guide and visualize the process of training and detection.
- Vtk 5.8 - Visualization Toolkit [Vtk14]  
Vtk provides a high level programming interface for implementing scientific visualizations. In the implementation to this thesis, Vtk has been used for visualizing 3D data.
- OpenMP - Open Multi-Processing [Ope14]  
Current state-of-the-art computers incorporate multi-core processors. Traditional sequential application development, however, only makes use of one processor at a time. OpenMP provides simple means to annotate the code with multi processing statements. The annotated code fragments are automatically run on different threads, based on the number of available CPU cores. Time critical algorithms can therefore be speeded up and optimized to work at best on the current system.
- ROS Hydromedusa - Robot Operating System [QGC<sup>+</sup>] [ROS14]  
Since the presented algorithm is supposed to be used in conjunction with the service robot LISA, ROS has been used to provide an interface.
- Libjsoncpp 0.6 [Jso14]  
In order to save trained data and be able to manipulate different parameters of the algorithm without the need to recompile the code, the JSON data exchange format has been chosen, since it provides human-readable means to storing data. The implementation uses libjsoncpp.
- Python 2.7.4 [Pyt14]  
Python is a scripting programming language and has been used in the context of evaluation, in order to run test cases and process the generated data.



# Bibliography

- [ABCO<sup>+</sup>03] ALEXA, Marc ; BEHR, Johannes ; COHEN-OR, Daniel ; FLEISHMAN, Shachar ; LEVIN, David ; SILVA, Claudio T.: Computing and Rendering Point Set Surfaces. In: *IEEE Transactions on Visualization and Computer Graphics* (2003)
- [AR02] AGARWAL, Shivani ; ROTH, Dan: Learning a Sparse Representation for Object Detection. In: *ECCV (4)*, 2002, S. 113–130
- [AS09] ALBALATE, Amparo ; SUENDERMANN, David: A Combination Approach to Cluster Validation Based on Statistical Quantiles. In: *IJCBS*, 2009, S. 549–555
- [Bal81] BALLARD, D. H.: Generalizing the Hough transform to detect arbitrary shapes. In: *Pattern Recognition* 13 (1981), Nr. 2, S. 111–122
- [Ben75] BENTLEY, Jon L.: Multidimensional binary search trees used for associative searching. In: *Communications of the ACM* 18 (1975), Nr. 9, S. 509–517
- [BHP01] BAREQUET, Gill ; HAR-PELED, Sariel: Efficiently Approximating the Minimum-Volume Bounding Box of a Point Set in Three Dimensions. In: *Journal of Algorithms* 38 (2001), Nr. 1, S. 91–109
- [Boo14] *Boost C++ Libraries*. <http://www.boost.org/>. Version: Februar 2014
- [BTG06] BAY, Herbert ; TUYTELAARS, Tinne ; GOOL, Luc J. V.: SURF: Speeded Up Robust Features. In: *ECCV* (2006), S. 404–417
- [CDF<sup>+</sup>04] CSURKA, Gabriella ; DANCE, Christopher R. ; FAN, Lixin ; WILLAMOWSKI, Jutta ; BRAY, Cédric: Visual categorization with Bags of Keypoints. In: *In Workshop on Statistical Learning in Computer Vision, ECCV*, 2004, S. 1–22

- [Che95] CHENG, Yizong: Mean Shift, Mode Seeking, and Clustering. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17 (1995), Nr. 8, S. 790–799
- [CM02] COMANICIU, D. ; MEER, P.: Mean Shift: A Robust Approach Toward Feature Space Analysis. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24 (2002), Nr. 5, S. 603–619
- [Com14] COMPUTER VISION LAB, DISI, UNIVERSITY OF BOLOGNA: *SHOT: Unique Signatures of Histograms for Local Surface Description*. <http://vision.deis.unibo.it/research/78-cvlab/80-shot>. Version: Februar 2014
- [CRM11] COMANICIU, Dorin ; RAMESH, Visvanathan ; MEER, Peter: The Variable Bandwidth Mean Shift and Data-Driven Scale Selection. In: *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, 2001
- [CV95] CORTES, Corinna ; VAPNIK, Vladimir: Support-Vector Networks. In: *Machine Learning* 20 (1995), Nr. 3, S. 273–297
- [DLR77] DEMPSTER, A. P. ; LAIRD, N. M. ; RUBIN, D. B.: Maximum likelihood from incomplete data via the EM algorithm. In: *Journal of the Royal Statistical Society: Series B* 39 (1977), Nr. 1, S. 1–38
- [DP97] DOMINGOS, Pedro ; PAZZANI, Michael: On the Optimality of the Simple Bayesian Classifier Under Zero-One Loss. In: *Mach. Learn.* 29 (1997), Nr. 2-3, S. 103–130
- [Eig14] *Eigen*. <http://eigen.tuxfamily.org/>. Version: Februar 2014
- [ER80] ECKES, Thomas ; ROSSBACH, Helmut: *Clusteranalysen*. Kohlhammer, 1980 (Kohlhammer Standards Psychologie : Studententext : T)
- [FH75] FUKUNAGA, Keinosuke ; HOSTETLER, Larry D.: The Estimation of the Gradient of a Density Function, with Applications in Pattern Recognition. In: *IEEE Transactions on Information Theory* 21 (1975), Nr. 1, S. 32–40
- [HA62] HOUGH, P. V. C. ; ARBOR, A.: Method and Means for Recognizing Complex Patterns / US Patent. 1962 (US Patent 3069 654). – Forschungsbericht
- [Har75] HARTIGAN, John A.: *Clustering Algorithms*. John Wiley & Sons, Inc., 1975

- [HDD<sup>+</sup>92] HOPPE, Hughes ; DEROSE, Tony ; DUCHAMP, Tom ; MCDONALD, John ; STUETZLE, Werner: Surface Reconstruction from Unorganized Points. In: *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, 1992, S. 71–78
- [HHRB11] HOLZ, Dirk ; HOLZER, S. ; RUSU, R.B. ; BEHNKE, Sven: Real-Time Plane Segmentation using RGB-D Cameras. In: *Proc. of the 15th RoboCup International Symposium*, 2011
- [HP01] HAR-PELED, Sariel: A Practical Approach for Computing the Diameter of a Point Set. In: *Proceedings of the Seventeenth Annual Symposium on Computational Geometry*, 2001, S. 177–186
- [HRD<sup>+</sup>12] HOLZER, Stefan ; RUSU, Radu B. ; DIXON, M. ; GEDIKLI, Suat ; NAVAB, Nassir: Adaptive Neighborhood Selection for Real-Time Surface Normal Estimation from Organized Point Cloud Data Using Integral Images. In: *IROS*, 2012
- [HS88] HARRIS, Chris ; STEPHENS, Mike: A combined corner and edge detector. In: *Fourth Alvey Vision Conference*. Manchester, UK, 1988, S. 147–151
- [Joa98] JOACHIMS, Thorsten: Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In: *ECML*, 1998, S. 137–142
- [Jso14] *JsonCpp - JSON data form manipulation library*. <http://jsoncpp.sourceforge.net/>. Version: Februar 2014
- [KPVG10] KNOPP, Jan ; PRASAD, Mukta ; VAN GOOL, Luc: Orientation Invariant 3D Object Classification Using Hough Transform Based Methods. In: *Proceedings of the ACM Workshop on 3D Object Retrieval*, 2010 (3DOR '10), S. 15–20
- [KPW<sup>+</sup>10] KNOPP, Jan ; PRASAD, Mukta ; WILLEMS, Geert ; TIMOFTE, Radu ; VAN GOOL, Luc: Hough Transform and 3D SURF for Robust Three Dimensional Classification. In: *ECCV (6)*, 2010, S. 589–602
- [Leh86] LEHMANN, E.L. ; CASELLA, Geroge (Hrsg.) ; FIENBERG, Stephen (Hrsg.) ; OLKIN, Ingram (Hrsg.): *Testing Statistical Hypotheses*. Springer-Verlag New York Berlin Heidelberg, 1986 (Springer Texts in Statistics)

- [Llo82] LLOYD, Stuart P.: Least squares quantization in pcm. In: *IEEE Transactions on Information Theory* (1982)
- [LLS04] LEIBE, Bastian ; LEONARDIS, Ales ; SCHIELE, Bernt: Combined Object Categorization and Segmentation With An Implicit Shape Model. In: *ECCV' 04 Workshop on Statistical Learning in Computer Vision*, 2004, S. 17–32
- [LLS06] LEIBE, Bastian ; LEONARDIS, Ales ; SCHIELE, Bernt: An Implicit Shape Model for Combined Object Categorization and Segmentation. In: *Toward Category-Level Object Recognition*, 2006, S. 508–524
- [LLS08] LEIBE, Bastian ; LEONARDIS, Ales ; SCHIELE, Bernt: Robust Object Detection with Interleaved Categorization and Segmentation. In: *International Journal of Computer Vision* 77 (2008), Nr. 1-3, S. 259–289
- [Low99] LOWE, David G.: Object Recognition from Local Scale-Invariant Features. In: *Proceedings of the International Conference on Computer Vision*. Corfu Greece, 1999, S. 1150–1157
- [LS03] LEIBE, Bastian ; SCHIELE, Bernt: Interleaved Object Categorization and Segmentation. In: *BMVC*, 2003
- [MCCO07] MARKLEY, F. L. ; CHENG, Yang ; CRASSIDIS, John L. ; OSHMAN, Yaakov: Quaternion Averaging. In: *Journal of Guidance Control and Dynamics* 30 (2007), Nr. 4, S. 1193–1197
- [MKB79] MARDIA, Kantilal V. ; KENT, John T. ; BIBBY, John M.: *Multivariate analysis*. Academic Press, 1979 (Probability and mathematical statistics)
- [MS02] MIKOLAJCZYK, Krystian ; SCHMID, Cordelia: An Affine Invariant Interest Point Detector. In: *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part I*. London, UK : Springer-Verlag, 2002. – ISBN 3–540–43745– 2, S. 128–142
- [Müt13] MÜTZEL, Andreas: *A Pose-Graph SLAM Frontend Based on Geometric Features*, Universität Koblenz-Landau, Diplomarbeit, 2013
- [NIH+11] NEWCOMBE, Richard A. ; IZADI, Shahram ; HILLIGES, Otmar ; MOLYNEAUX, David ; KIM, David ; DAVISON, Andrew J. ; KOHLI, Pushmeet ; SHOTTON, Jamie ; HODGES, Steve ; FITZGIBBON, Andrew: KinectFusion: Real-time Dense Surface Mapping and Tracking.



- In: *Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality*, 2011, S. 127–136
- [NMTM98] NIGAM, Kamal ; MCCALLUM, Andrew ; THRUN, Sebastian ; MITCHELL, Tom M.: Learning to Classify Text from Labeled and Unlabeled Documents. In: *AAAI/IAAI*, 1998, S. 792–799
- [Ope14] *OpenMP.org - The OpenMP API Specification for Parallel Programming*. <http://openmp.org/>. Version: Februar 2014
- [PASW13] PAPON, Jeremie ; ABRAMOV, Alexey ; SCHOELER, Markus ; WÖRGÖTTER, Florentin: Voxel Cloud Connectivity Segmentation - Supervoxels for Point Clouds. In: *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, 2013
- [PM00] PELLEG, Dan ; MOORE, Andrew: X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In: *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000, S. 727–734
- [Poi14] *Point Cloud Library (PCL)*. <http://pointclouds.org/>. Version: Februar 2014
- [Pyt14] *Python Programming Language*. <http://www.python.org/>. Version: Februar 2014
- [QGC<sup>+</sup>] QUIGLEY, Morgan ; GERKEY, Brian ; CONLEY, Ken ; FAUST, Josh ; FOOTE, Tully ; LEIBS, Jeremy ; BERGER, Eric ; WHEELER, Rob ; NG, Andrew: *ROS: an open-source Robot Operating System*
- [QtP14] *Qt Project*. <http://qt-project.org/>. Version: Februar 2014
- [RBB09] RUSU, Radu B. ; BLODOW, Nico ; BEETZ, Michael: Fast point feature histograms (FPFH) for 3D registration. In: *Proc. of the International Conference on Robotics and Automation (ICRA)*. IEEE, 2009, S. 3212–3217
- [RC11] RUSU, Radu B. ; COUSINS, Steve: 3D is here: Point Cloud Library (PCL). In: *Proc. of the 2011 IEEE International Conference on Robotics and Automation (ICRA)*, 2011, S. 1–4
- [RMBB08] RUSU, Radu B. ; MARTON, Zoltan C. ; BLODOW, Nico ; BEETZ, Michael: Persistent Point Feature Histograms for 3D Point Clouds. In: *Proceedings of the 10th International Conference on Intelligent Autonomous Systems*, 2008

- [ROS14] *ROS.org / Powering the world's robots.* <http://www.ros.org/>.  
Version: Februar 2014
- [Rus09] RUSU, Radu B.: *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*, Computer Science department, Technische Universitaet Muenchen, Germany, Diss., 2009
- [SAS07] SCOVANNER, Paul ; ALI, Saad ; SHAH, Mubarak: A 3-dimensional Sift Descriptor and Its Application to Action Recognition. In: *Proceedings of the 15th International Conference on Multimedia*, 2007, S. 357–360
- [SKM<sup>+</sup>13] SEIB, Viktor ; KATHE, Florian ; MCSTAY, Daniel ; MANTHE, Stephan ; PETERS, Arne ; JÖBGEN, Benedikt ; MEMMESHEIMER, Raphael ; JAKOWLEWA, Tatjana ; VIEWEG, Caroline ; STÜMPER, Sebastian ; GÜNTHER, Sebastian ; MÜLLER, Simon ; VEITH, Alruna ; KUSENBACH, Michael ; KNAUF, Malte ; PAULUS, Dietrich: RoboCup 2013 - homer@UniKoblenz (Germany) / Universität Koblenz-Landau, www.uni-koblenz.de. 2013. – Forschungsbericht
- [Sta14] STANFORD UNIVERSITY COMPUTER GRAPHICS LABORATORY: *Stanford 3D Scanning Repository.* <http://graphics.stanford.edu/data/3Dscanrep/>. Version: Februar 2014
- [STDS10] SALTI, Samuele ; TOMBARI, Federico ; DI STEFANO, Luigi: On the Use of Implicit Shape Models for Recognition of Object Categories in 3D Data. In: *ACCV (3)*, 2010 (Lecture Notes in Computer Science), S. 653–666
- [TSDS10] TOMBARI, Federico ; SALTI, Samuele ; DI STEFANO, Luigi: Unique signatures of histograms for local surface description. In: *Proc. of the European conference on computer vision (ECCV)*. Berlin, Heidelberg : Springer-Verlag, 2010 (ECCV'10). – ISBN 3–642–15557– X, 978–3–642–15557–4, S. 356–369
- [TSDS11] TOMBARI, Federico ; SALTI, Samuele ; DI STEFANO, Luigi: A combined texture-shape descriptor for enhanced 3d feature matching. In: *Proc. of the International Conference on Image Processing (ICIP)* IEEE, 2011, S. 809–812
- [Vap99] VAPNIK, Vladimir ; VAPNIK, Vladimir (Hrsg.): *The Nature of Statistical Learning Theory*. Springer-Verlag, 1999

- [Vtk14] *Vtk - The Visualization Toolkit*. <http://www.vtk.org/>.  
Version: Februar 2014
- [Zan07] ZANT, Tijn van d.: RoboCup@Home: Creating and Benchmarking Tomorrows Service Robot Applications. In: *Robotic Soccer (2007)*, S. 521–528
- [Zho09] ZHONG, Yu: Intrinsic shape signatures: A shape descriptor for 3D object recognition. In: *2009 IEEE 12th International Conference on Computer Vision workshops, ICCV*, 2009, S. 689–696