TECHNISCHE UNIVERSITÄT MÜNCHEN

Fakultät für Informatik
Computer Aided Medical Procedures & Augmented Reality / I16

# Point Cloud Computing for Rigid and Deformable 3D Object Recognition

Bertram Heinrich Drost

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

| | | |
|---|---|---|
| Vorsitzender: | | Univ.-Prof. Dr.-Ing. Darius Burschka |
| Prüfer der Dissertation: | | |
| | 1. | Univ.-Prof. Dr. Nassir Navab |
| | 2. | Univ.-Prof. Dr. Vincent Lepetit, TU Graz |

Die Dissertation wurde am 21. Juni 2016 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 1. November 2016 angenommen.

**Abstract**

Machine vision is a technologically and economically important field of computer vision. It eases automatization of inspection and manipulation tasks, which in turn enables cost savings and quality improvement in industrial processes. Usually, 2D or intensity images are used for such applications. However, thanks to several technological advances, nowadays there are sensors available that allow depth or 3D measurements with high resolution, frequency and accuracy at a reasonable cost. Such 3D data enables new applications that are difficult or impossible to implement with 2D images only.

This work develops several performant, robust and accurate algorithms for processing such 3D data. The algorithms were developed with the requirements of industrial image processing in mind. They are, however, applicable to other areas such as robotics and reverse engineering as well. Two fundamental challenges are solved in this work: The fast localization of 3D points that neighbor a given query point and the detection of rigid and deformable objects in 3D point clouds or in multimodal data. Additionally, a fast and robust method for refining the position of two point clouds is presented and some fundamental algorithms regarding rotations are discussed.

For the detection of nearest neighbors in 3D point clouds, a voxel-based method is introduced that allows almost constant lookup times of $\mathcal{O}(log(log(N)))$. Additionally, in contrast to prior art, lookup times are almost independent of the distribution of query and data points, allowing the use of this method in real-time systems. A variant of the method that allows for approximate nearest neighbor lookups further improves the required time for creating the underlying data structure.

For the detection of rigid and deformable objects in 3D and multimodal data, a local variant of the Hough transform is introduced that circumvents the usual problems of voting schemes over high-dimensional parameter spaces. The very robust, fast, and generic baseline method detects rigid objects in 3D point clouds. It works for arbitrary free-form objects and different 3D sensors alike and can cope with large amounts of clutter, noise, occlusion, sparse data and multiple object instances.

Three variants of the baseline method are introduced that detect rigid objects in multimodal data, geometric primitives in 3D point clouds, and deformable objects in 3D point clouds. For this, the parameter space, the used feature, the model representation and the number of voting rounds of the original method are modified according to the type of data and model. All methods are evaluated using up-to-date datasets.

## Zusammenfassung

Ein technologisch und wirtschaftlich wichtiges Anwendungsgebiet der computergestützten Bildverarbeitung ist die industrielle Bildverarbeitung. Diese vereinfacht vielfach die Automatisierung industrieller Inspektions- und Manipulationsaufgaben, was Kosteneinsparungen und Qualitätsverbesserungen ermöglicht. Klassischerweise werden dafür Intensitäts- oder 2D-Bilder verwendet. Dank diverser technologischer Fortschritte sind heute allerdings auch preiswerte Sensoren verfügbar, die Tiefen- oder 3D-Aufnahmen mit hoher Auflösung, Frequenz und Genauigkeit liefern. Derartige 3D-Daten ermöglichen Anwendungen die mit 2D-Bildern nicht oder nur schwer lösbar sind.

Diese Arbeit entwickelt mehrere schnelle, robuste und genaue Verfahren zur Verarbeitung derartiger 3D-Daten. Die Verfahren wurden in Hinblick auf typische Anforderungen der industriellen Bildverarbeitung entwickelt, sind aber auch in anderen Bereichen wie der Robotik oder dem Reverse Engineering einsetzbar. Zwei Kernprobleme werden dabei gelöst: Das schnelle Finden von Punkten, die in der Nähe eines gegebenen Punktes liegen, sowie die Lagebestimmung von starren und deformierbaren Objekten in 3D bzw. multimodalen Daten. Zusätzlich wird ein robustes und schnelles Verfahren zur Verfeinerung der Lage zweier Punktwolken vorgestellt sowie einige algorithmische Grundlagen zu Rotationen aufgearbeitet.

Für das Auffinden nächster Nachbarn in 3D-Punktwolken wird ein Voxelbasiertes Verfahren vorgestellt, welches ein annähernd konstante Suchzeit von $\mathcal{O}(log(log(N)))$ ermöglicht, die zusätzlich, anders als frühere Ansätze, größtenteils unabhängig von der Verteilung der Daten- und Suchpunkte ist. Eine darauf aufbauende approximative Suche ermöglicht zusätzlich ein schnelles generieren der zugrundeliegenden Datenstrukturen.

Für die Lagebestimmung starrer und deformierbarer Objekte in 3D und multimodalen Daten wird eine Variation der Hough-Transformation vorgestellt, welche die üblichen Fallstricke dieser Methode bei hochdimensionalen Parameterräumen durch eine lokale, datengetriebene Parametrisierung der Objektlage umgeht. Das äußerst robuste, schnelle und allgemeine Basisverfahren findet starre Objekte in 3D-Punktwolken. Es funktioniert für beliebige Freiformflächen und verschiedenste 3D-Sensoren und kann mit großen Mengen an Störpunkten, Rauschen, Verdeckungen, dünnen Daten und mehreren Objektinstanzen umgehen.

Vom Basisverfahren ausgehend werden weitere Verfahren für die Detektion von starren Objekten in multimodalen Daten, von geometrischen Primitiven in 3D-Punktwolken sowie von deformierbaren Objekten in 3D-Punktwolken vorgestellt. Dazu werden der Parameterraum, die verwendeten Merkmale, die Modelldarstellung sowie die Anzahl der Abstimmungsrunden des Verfahrens entsprechend der Daten und des Modells modifiziert. Alle vorgestellten Verfahren werden anhand aktueller Datensätze evaluiert.

**Acknowledgments**

This work would not have been possible without the help and support of many people, whom I have the pleasure to thank here.

At CAMPAR, the chair I was lucky to be a part of, Dr. Slobodan Ilic for his extensive and valuable guidance over the years and countless fruitful discussions. Prof. Dr. Nassir Navab for his inspiration, motivation and for creating and encouraging the collaborative and productive atmosphere at CAMPAR. All other members of the vision group - Alexander, Benoit, Cedric, Christian, Danilo, David, Federico, Paul, Sebastian, Stefan$^2$, Tolga, Vasilis, Vladimir, and many more - for all their support.

At MVTec, who supported this work, Prof. Carsten Steger for seeing the opportunity of 3D machine vision and for initiating this project. Dr. Markus Ulrich for his support and advice. And all other colleagues at MVTec who were more than helpful along the way, especially Andreas for paving the way.

Finally, all of my family, who always stood behind me. Christine, for waiting patiently while I was busy. Astrid, for peanut-proofing papers. My parents, on whose shoulders I stand. And most importantly Georgiana, for her never-ending love, motivation, and encouragement – and patience, when I was running in circles when late-night deadlines approached. This would not have been possible without you.

# Contents

**Outline of the Thesis**

## Part I: Introduction and Basics

Chapter 1: Introduction

motivates this work and gives an overview of the challenges and solutions.

Chapter 2: Notations and Fundamentals

gives an overview of the used notation and introduces some of the basic mathematical concepts and algorithms used in this work.

## Part II: Nearest Neighbors and Pose Refinement in 3D

of this thesis presents and explores two fundamental algorithms which stand for themself.

Chapter 3: Voxel-Hash Based Nearest Neighbor Search

introduces a novel method for solving the nearest-neighbor problem in 3D. It uses an octree-based discretization of space that is built on top of a Voronoi tesselation of the input points and optimizes access to the octree using a hash table. This results in an almost constant lookup time. An approximate variant of the nearest neighbor scheme is presented which is faster for both lookup and data structure creation.

Chapter 4: A Variant of the Iterative Closest Points Algorithm

presents a variant of the iterative closest points (ICP) method, an algorithm that optimizes a given rigid transformation between two 3D point clouds by minimizing the distances between the clouds. It discusses in detail the design choices and also presents a variant that optimizes the poses of multiple point clouds w.r.t. each other simultaneously.

## Part III: Matching in 3D Point Clouds

presents four methods for detecting and localizing 3D objects in 3D data. The methods differ in the type of objects they detect (free-form vs. primitives), the type of transformations they recover (rigid vs. deformable) and the type of data they use (3D only vs. multimodal). All four methods are based on a common baseline method which is presented in the first of the four chapters.

Chapter 5: Rigid Object Detection in 3D Point Clouds: A Local Voting Scheme

presents a method for detecting rigid 3D objects in 3D point clouds, using a Hough-transform like voting scheme that operates on a data-driven, local parameter space. It uses features that describe pairs of 3D points in an invariant way. This method is the baseline method for the subsequent chapters, which modify the voting scheme in one or more aspects. It is therefore longer than the others and explores the aspects of the voting scheme in more detail.

Chapter 6: Rigid Object Detection in Multimodal Data

presents a method for detecting rigid 3D objects in multimodal RGB-D data. Different from the baseline method, it uses a novel multimodal point pair descriptor that combines invariant and robust information from both modalities.

Chapter 7: Primitive Shape Detection in 3D Point Clouds

shows how the baseline method can be modified to detect primitive, symmetric shapes (planes, spheres and cylinders) in 3D point clouds. Compared to the free-form detector, the symmetries of the shapes are used to both reduce the parameter space and to optimize the feature matching.

Chapter 8: Deformable 3D Object Detection in 3D Point Clouds

presents a way of finding deformable 3D objects in 3D point clouds. It embedds the voting scheme into a graph matching framework to performs an iterative, re-weighted voting. Based on a few example deformations of the object, the deformed point pair features are trained, allowing the detection of model-free deformations.

# Part I

# Introduction and Basics

# 1 Introduction

## 1.1 Background and Motivation

**Machine Vision** Ever since the dawn of electronic image capturing devices, machine vision is a driving factor behind the increased automation of industrial processes. As part of automated feedback control systems, it allows to both reduce labor costs and raise the overall product quality, up to high-volume zero-defect production chains. In the broadest sense, machine vision systems act as sensors in feedback control systems. More specifically, machine vision allows to solve a wide variety of control, inspection and manipulation tasks. Thanks to a large array of different components for lighting, lenses, sensors, and software and thanks to being contact-less, it is applicable to a wide range of industrial scenarios [132].

The advantages of using Machine Vision for different tasks are numerous. Overall and in the long term, it reduces costs and raises quality. On a more detailed level, machine vision

- allows early detection of departures from given tolerances, which in turn allows both immediate correction of processing parameters as well as cost-saving removals of defective parts early in the production process;
- is able to operate consistently around the clock, avoiding unnoticed mistakes due to, for example, lack of concentration;
- can be fast, sometimes allowing the inspection of thousands of parts per second;
- is often cheaper than human labor in the long run, an advantage that increases with falling hardware and raising labor costs;[1]

[1]The economic and social implication of increased automation are beyond the scope of this work. In the short term, automation of processes often leads to job loss and sometimes social unrest. In the long term and more positively, it can free people from exhausting or dangerous production environments, increases

- is able to measure orders of magnitude more accurately than the human eye; and
- allows for the perception of microscopic structures that are beyond the capabilities of humans.

Machine vision is closely related to *computer vision*, and the two fields often overlap. Compared to computer vision, machine vision applications often have stricter requirements regarding processing speed, accuracy, robustness and ease of configuration. This is feasible because machine vision applications are typically deployed in more controlled environments that are optimized regarding the given task and that show little variation over time. Environment and system parameters such as lighting, exposure time, focal length, aperture settings, working distance and others are optimized for the particular setup and often remain unchanged over the lifetime of the system.

**3D Machine Vision** Traditional machine vision is to a large degree based on the processing of 2D images [132]. This includes the processing of 3D data, which is often acquired and processed in the form of range images, for example by thresholding heights or comparing to reference range images. Even though such image-based algorithms work for a wide range of applications, full-fledged, non-image-based 3D methods that directly process 3D point clouds, 3D meshes or other kind of 3D data allow for an even wider range of challenges to be solved. For example, localizing free-form texture-less objects is significantly more difficult when using 2D images only, as is finding defects on the surface of such free-form objects.

Over the last four decades, advances in microelectronics lead to systems that today are over a million times faster at comparable prices [94]. Additionally, especially over the last decade, advances in electronics lead to 3D sensors and acquisition devices that are more available, more robust, more accurate and less expensive. Today, one can choose from a multitude of different 3D input devices that cover a large range of different characteristics, such as time-of-flight sensors, radar and LIDAR scanners, precalibrated stereo systems working in the visible or infrared spectrum, or structured light sensors.

This thesis tackles one of the remaining reasons for the lack of wide-spread 3D machine vision: The lack of generic 3D data and point cloud processing algorithms that fulfill the industrial requirements and that are able to solve the typical challenges arising in industrial setups.

productivity, thus generating economical growth, and is often a key for mass-producing complex technology which thus becomes cheaper and more available. Nevertheless, the past has seen multiple riots by workers who were replaced by machines. The first were probably weavers who were replaced on large scales by the semiautomatic *power looms* in the beginning of the 19th century. [143, 33]

**Acquisition Methods** Numerous sensors, devices and methods for acquiring 3D data are available today, and many more were proposed in the literature. The particular method for a setup is usually chosen depending on the specific application's requirements, such as the required accuracy, cycle times, object and scene dimensions, object and surface characteristics, costs, complexity and ease of use, mounting restrictions, environmental conditions, and others.

Apart from those characteristics, 3D acquisition methods can also be categorized by the underlying physical method that is used. Methods based on electromagnetic waves include active and passive triangulation (such as stereo, structure from motion, sheet-of-light, and structured light), phase shift and signal runtime measurements, depth from (de)focus, depth from shading, photometric stereo, and others.

Beside electromagnetic waves, acoustic signals can be used to obtain measurements of a scene. Bats use ultrasound principle to navigate in dark caves as well as to locate prey.[2] Sound waves are used in the form of active and passive sonars by ships and submarines to measure the ground and to locate floating objects. Medical ultrasound allows imaging and measuring of the inside of organic bodies.

Because of that large variety of different methods, the data of the corresponding sensors have a large variety of different characteristics which in turn lead to different algorithmic challenges. Such challenges include in particular

- **Noise** that disturbs the measured points, with levels that range from very low volume to noise ratios (structured light, sheet-of-light with micrometer accuracy) to very high ratios (time-of-light sensors with noise of up to 1 cm).
- **Veil points**, where at depth discontinuities, points between the two depth levels are returned that are not actually part of the scene.
- **Noise points**, which are random points within the measurement volume that are not actually part of the scene.
- **Missing data**, where parts of the scene are not reconstructed. For example, stereo sensors might leave out areas without texture; also, time-of-flight sensors often fail to reconstruct objects with certain surface materials.

[2]The echolocation capabilities of bats are amazing. Some bats can detect wires as thin as 0.3 mm from a distance of 2 m. Their prey includes up to 1200 fruit flies per day, which are around 3 mm long. Bats combine constant frequency and modulated frequency methods, using the Doppler effect, runtime differences as low as 0.1 ms between both ears, and likely the inference of the returned sound waves to obtain distances and relative movements of their surroundings. Additionally, they use the loudness of the returned waves to gain information about the surface characteristics and the material. Since the runtime of sound is temperature dependent, the bat's echolocation incorporates the ambient temperature. Prey is additionally identified by its flap frequency, which is obtained by analyzing the frequency shift due to the Doppler effect at the preys wings. An evolutionary countermeasure includes moths that stop flapping when detecting corresponding ultrasound pulses. [142]

- **Low resolution**: While some sensors can have resolutions in the range of megapixels, others (such as certain time-of-flight sensors) return only some hundred to thousand points, leading to more sparse reconstructions.
- **Representation**: Many methods reconstruct only points that are visible from a particular single viewpoint, such as the camera or sensor center. For such sensors, the 3D data can be represented in the form of range, depth or XYZ images. Such a representation has the advantage that quite often, the neighborhood structure of the pixels can be exploited as neighboring pixels in the image often represent points that are neighboring in 3D. However, other methods reconstruct data from multiple viewpoints and return a 3D point cloud for which no image-based representation exists.
- **Multimodal data**: Some sensors return additional data such as intensity or RGB images in varying quality, which algorithms can exploit.

In order to maximize the range of possible applications and thus their economic impact, algorithms that process 3D data should work with data from as many different sources as possible. In particular, such methods should be as robust as possible against noise, clutter, missing data, should work with sparse data and operate on 3D point clouds instead of range or depth images.

## 1.2 Objectives

The overall objective of this work was to develop methods that allow solving a wide variety of industrial and robotic applications in 3D. Such solutions require different algorithmic building blocks, each of which should fulfill several requirements in order to be applicable in as many scenarios as possible and to thus maximize their economic impact.

In a nutshell, the aim of this work was to develop 3D object detection algorithms and 3D surface comparison methods that are generic, fast, robust, accurate, and easy to use.

**Application Scenarios** Many industrial applications fall into one of two categories *inspection* and *manipulation* [132, 9]. Here, inspection denotes measurements performed in the acquired data that are a basis for subsequent decisions. Such measurements allow, for example, detecting defective parts or adjusting process parameters using feedback control systems. Inspection of an object often requires that the object is localized[3] in the scene. In other words, its presence and exact position in the scene must be determined. After alignment, comparisons to a reference model or other measurements that are defined in model coordinates

[3]Localization is also sometimes called *object detection*, *object registration*, *object recognition*, or *position detection*, terms this work will use interchangeably.

can be performed to check if the part fulfills required tolerances or has required characteristics.

Manipulation is any physical handling of an object, such as picking it up with a robot, cutting it, drilling into it, or assembling two or more parts. One very typical application is bin picking, where many instances of an object are arranged in a heap and a robot must pick up one that lies on top. As for inspection, a first step for many manipulation tasks is the localization of the target object, as it allows to transfer locations defined in model coordinates (such as gripping points) into scene coordinates. Note that inspection and manipulation can be combined, for example, to remove a part after it was deemed defective.

**Required Algorithms** Both inspection and manipulation thus often require a localization of the target object in the scene. Even though this step can be avoided by using a physical setup that puts objects repeatedly at the exact same position, this is often not feasible or more expensive than doing the alignment in software. One of the most important algorithms required for industrial 3D machine vision is thus a 3D object detector that establishes if a particular object is present in the scene and, if present, accurately localizes all instances of the object. Consequently, the main objective of this work was to create such a detector. As outlined in more detail in Sec. 5, while such methods exist in the literature, they are not as generic, robust and efficient as required by industrial applications. Sec. 5 through Sec. 8 will develop a novel object detection framework and apply it to objects, data sources, and transformations of different kind.

The object detector developed later obtains the object's position only up to a certain degree of accuracy because of several approximations used throughout the method. In order to further improve that approximate position, a robust local refinement method is required, which is described in Sec. 4.

Another important algorithmic building block, required to perform point-to-point surface comparisons and to efficiently perform the 3D pose refinement mentioned above, is a fast 3D nearest neighbor lookup method that, given a 3D query point and a 3D data point cloud, quickly determines the data point that is closest to the query point. We will see that while such methods already exists, they are not as efficient as possible and their runtime often depends on the distribution of the data and query points. Sec. 3 will develop a 3D nearest neighbor lookup method that overcomes these drawbacks.

**Algorithm Requirements** As briefly mentioned above, the methods employed in machine vision applications should fulfill certain requirements in order to maximize their range of possible applications, the main requirements being that the methods are generic, fast enough, robust, accurate, and easy to use.

Industrial objects come in many shapes. *Generality* requires the methods to work with as many different kinds of objects as possible. The methods should especially not be limited to special classes such as primitive shapes. As mentioned above, generality also means that the method must work with a large range of different acquisition methods that exhibit different characteristics in terms of noise, accuracy, missing data, density, and data representation. Generality essentially makes the difference between a hand-written algorithm for a particular task and a generic method that can be part of a wide variety of setups.

As part of the generality, the method must be *easy to use*. This requirement translates to the condition that it must not have too many parameters that the user has to set. If parameters are necessary, they should either be computed automatically or be reasonably intuitive. Also, the amount of required manual preprocessing should be minimal.

For practical applications, the overall pipeline from scene acquisition to the final pose must be *reasonably fast*. While the exact cycle times vary between industries and products, one must consider that those times apply to the whole cycle of acquisition, image processing and actuation. The faster the individual steps are, the more applications can be covered.

*Robustness* is a general term that represents the expectation that the key properties of the methods – such as speed, accuracy and detection rate – must not change drastically with small variations of the input, such as noise, unexpected data or unexpected deformations of the object.

Finally, the method must be *accurate* enough to allow the user to finish the task at hand. Similar to the condition of being as fast as possible, different applications have different requirements regarding accuracy: while picking up an object with a vacuum actuator might not require a very high accuracy, comparing surfaces require poses that are at least as accurate as the surface variation that shall be detected. In general, one expects the method to at least not be less accurate than the input data provided by the sensor.

## 1.3 Publications

Parts of this thesis contain material previously published in the following publications. To date, the first publication was cited over 180 times.

- BERTRAM DROST, MARKUS ULRICH, NASSIR NAVAB, AND SLOBODAN ILIC, *Model globally, match locally: Efficient and robust 3D object recognition*, in The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010, IEEE Computer Society, 2010, pp. 998–1005

- BERTRAM DROST AND SLOBODAN ILIC, *3D Object Detection and Localization Using Multimodal Point Pair Features*, in 2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission, Zurich, Switzerland, October 13-15, 2012, IEEE Computer Society, 2012, pp. 9–16

- Bertram Drost and Slobodan Ilic, *A Hierarchical Voxel Hash for Fast 3D Nearest Neighbor Lookup*, in Pattern Recognition - 35th German Conference, GCPR 2013, Saarbrücken, Germany, September 3–6, 2013. Proceedings, Joachim Weickert, Matthias Hein, and Bernt Schiele, eds., vol. 8142 of Lecture Notes in Computer Science, Springer, 2013, pp. 302–312
- Bertram Drost and Slobodan Ilic, *Graph-Based Deformable 3D Object Matching*, in Pattern Recognition - 37th German Conference, GCPR 2015, Aachen, Germany, October 7-10, 2015, Proceedings, Juergen Gall, Peter V. Gehler, and Bastian Leibe, eds., vol. 9358 of Lecture Notes in Computer Science, Springer, 2015, pp. 222–233
- Bertram Drost and Slobodan Ilic, *Local Hough Transform for 3D Primitive Detection*, in 2015 International Conference on 3D Vision, 3DV 2015, Lyon, France, October 19-22, 2015, Michael S. Brown, Jana Kosecká, and Christian Theobalt, eds., IEEE, 2015, pp. 398–406

Parts of this thesis contain material previously published in the following patents.

- Bertram Heinrich Drost and Markus Ulrich, *Recognition And Pose Determination Of 3d Objects In 3d Scenes Using Geometric Point Pair Descriptors And The Generalized Hough Transform*, 2010. EP 2385483
- Bertram Drost and Markus Ulrich, *Recognition and pose determination of 3D objects in multimodal scenes*, 2012. EP 2720171

# 2
# Notations and Fundamentals

This chapter introduces some of the basic concepts, nomenclature and notations used in this work. Readers familiar with Euclidean geometry in 3D, the rotation group SO(3) and its parametrizations may wish to skip this chapter and refer to Fig. 2.1 for a brief summary of the notations.

## 2.1 3D Rotations

One practical issue that frequently arises when dealing with 3D transformations is the parametrization of 3D rotations. The properties of the group of 3D rotations is significantly more complex than in 2D, where a single angle is enough to describe each rotation. Different algorithms have different requirements, especially regarding computational costs, numerical robustness, uniqueness of the parametrization, and singularities in the parametrization.

Euler has shown in 1776 that the group of rotations in three-dimensional Eucledian space is a three-dimensional manifold [137]. As such, parametrizing 3D rotations requires at least three parameters.[1] However, as shown by Stuelpnagel in 1964 [137], every parametrization that uses a minimal set of exactly three parameters has at least one singular point within the rotation group. This is of disadvantage when computing derivatives w.r.t. the rotation parameters inside optimization methods. Hopf [70] showed that there are different compromises when choosing the number of parameters used to represent a 3D rotation. He showed that at least five parameters are necessary to represent the rotation group in a 1-1 global manner.

In this work, we generally use rotation matrices to represent rotations, as they have no singularity and represent rotations in a 1-1 manner. For optimizations, we parametrize an updating rotation using the Rodruiges parametrization, since it has exactly three parameters (thus avoiding overparametrization) and is smooth

[1]As in "three independent parameters". One might use using space-filling curves or similar techniques to compress three parameters into a single number.

Table 2.1: Notation used throughout this work.

| | |
|---|---|
| | **Spaces** |
| $\mathbb{R}$ | Set of real numbers |
| $\mathbb{R}^n$ | Vector space of real numbers with dimension $n$ |
| $\mathbb{Z}$ | Set of all integers |
| $\mathrm{SO}(3)$ | Special orthogonal group of dimension 3 that contains all 3D rotations |
| $\mathrm{SE}(3)$ | Special euclidean group of dimension 3 that contains all rigid 3D transformations |
| | **Vectors** |
| $\mathbf{x} \in \mathbb{R}^n$ | Column vector |
| $\mathbf{x}^T$ | Transposed vector |
| $\mathbf{n}(\mathbf{x})$ | Normal vector of point $\mathbf{x} \in \mathbb{R}^3$ |
| $\mathbf{a}_x$, $\mathbf{a}_y$, $\mathbf{a}_z$ | First, second, and third component of a three-dimensional vector $\mathbf{a} \in \mathbb{R}^3$ |
| | **Operators** |
| $\mathbf{a} \cdot \mathbf{b}$ | Euclidean scalar product, $\mathbf{a}^T\mathbf{b}$ |
| $\mathbf{a} \times \mathbf{b}$ | cross product |
| $\measuredangle(\mathbf{a}, \mathbf{b})$ | Angle between **a** and **b** (see Sec. 2.2.2) |
| $\measuredangle_{\mathbf{d}}(\mathbf{a}, \mathbf{b})$ | Signed angle between **a** and **b** w.r.t. **d** (see Sec. 2.2.3) |
| $\measuredangle R$ | Rotation angle of a rotation matrix $R \in \mathrm{SO}(3)$ |
| $R_{\mathbf{d}}(\alpha)$ | 3D rotation around axis **d** with an angle of $\alpha$ |
| $\mathrm{diam}(M)$ | Diameter of a point cloud $M \subset R^3$, $\mathrm{diam}(M) = \max\{\lvert\mathbf{x} - \mathbf{y}\rvert : \mathbf{x}, \mathbf{y} \in M\}$ |

Table 2.2: Overview over some common ways to parametrize the rotation group SO(3), their properties and the ease of use for certain problems. **Parameters** is the number of real-valued parameters. **Continuous** states if the parameters are continuous for small variations of the rotation. A **gimbal lock** is a situation where one degree of freedom is lost locally. The **ambiguity** states how many parametrizations exists for a particular rotation. Note that all parametrizations can be made unique by restricting the range of allowed parameters, at the loss of continuity. **Interpolate** states if there is a straightforward way to interpolate or to average over two or more rotations. **Compose**, **invert** and **apply** (*i.e.*, rotate a vector) state if the corresponding operations are easily performed. Finally, for some parametrizations there exists a **metric** which is easy to compute based on the parameters.

| | Parameters | Continuous | Gimbal Lock Free | Ambiguity | Complete Space | Interpolate | Compose | Invert | Apply | Metric |
|---|---|---|---|---|---|---|---|---|---|---|
| Axis + Angle | 4 | - | + | $\infty$ | - | - | - | + | + | - |
| Rodriguez | 3 | + | + | $\infty$ | + | - | - | + | + | - |
| Euler-Angles | 3 | + | - | $\infty$ | + | - | - | - | - | - |
| Quaternions | 4 | + | + | 2 | - | + | + | + | + | + |
| Rotation Matrix | 9 | + | + | 1 | - | + | + | + | + | + |

around the identity rotation, which is the domain where small updates are typically located.

### 2.1.1 Parametrizations

**Axis and Angle** As shown by Euler in 1775 [51], every 3D rotation can be represented by a (normalized) rotation axis $\mathbf{a} \in \mathbb{R}^3$, $|\mathbf{a}| = 1$ and a rotation angle $\phi$.[2] While this parametrization is rather unhandy for many operations, it has a well-defined physical interpretation. As such, it is good for visualizing or defining a rotation.

The so-called *Rodruiges' rotation formula* is used to rotate an arbitrary vector $\mathbf{v} \in \mathbb{R}^3$:

$$R(\mathbf{a}, \phi)\mathbf{v} = \mathbf{v}\cos(\phi) + (\mathbf{a} \times \mathbf{v})\sin(\phi) + \mathbf{a}(\mathbf{a} \cdot \mathbf{v})(1 - \cos(\phi)) \tag{2.1}$$

[2]Different geometric arguments lead to this theorem, and one consequence is the fact that when rotating a ball, there will always be at least two fixed points which oppose each other on the ball's surface. Perhaps unsurprising, the conjecture is called the *Satz vom Fußball* (soccer ball theorem) in german: When placing the soccer ball in the center of the field at the beginning of each half time, at least two points on the ball's surface will be at the same position.

where $R(\mathbf{a}, \phi)$ is the rotation around the normalized axis $\mathbf{a}$ with angle $\phi$.

If several vectors are to be rotated with the same axis and angle, it might be faster to precompute the corresponding rotation matrix. Using the convenient shortcuts $c = \cos(\phi)$ and $s = \sin(\phi)$, the matrix reads

$$R(\mathbf{a}, \phi) = \begin{pmatrix} c + \mathbf{a}_x^2(1-c) & \mathbf{a}_x\mathbf{a}_y(1-c) - \mathbf{a}_z s & \mathbf{a}_x\mathbf{a}_z(1-c) + \mathbf{a}_y s \\ \mathbf{a}_y\mathbf{a}_x(1-c) + \mathbf{a}_z s & c + \mathbf{a}_y^2(1-c) & \mathbf{a}_y\mathbf{a}_z(1-c) - \mathbf{a}_x s \\ \mathbf{a}_z\mathbf{a}_x(1-c) - \mathbf{a}_y s & \mathbf{a}_z\mathbf{a}_y(1-c) + \mathbf{a}_x s & c + \mathbf{a}_z^2(1-c) \end{pmatrix} \tag{2.2}$$

One exceptional case in the axis-angle-parametrization is the identity, for which the rotation angle is zero and the axis is an arbitrary unit vector. Because of this, the parametrization of the axis becomes unstable around the identity rotation: rotations that are (in SO(3)) arbitrarily close to identity, such as $\mathbf{a}_1 = (1, 0, 0)^T$, $\phi_1 = \epsilon$ and $\mathbf{a}_2 = (0, 1, 0), \phi_2 = \epsilon$, have very different axes. The parametrization of the axis is thus not continuous around the identity, leading to problems when, for example, computing partial derivatives.

**Rodruiges Parametrization** To avoid this problem of continuity, the *Rodrigues parametrization* of a 3D rotation uses the product of the angle $\phi$ and the rotation axis $\mathbf{a}$

$$\phi\mathbf{a} \tag{2.3}$$

to describe a 3D rotation [112]. Since the length of the vector approaches zero if the rotation approaches identity, this parametrization is continuous even around the identity rotation.

A distinctive advantage of the Rodrigues parametrization is that it is not overparameterized and has no singularity around the identity. This makes it suitable for parametrizing an update rotation for optimization processes. We thus use the Rodrigues parametrization for the optimizations in Sec. 4.

A variant of the Rodruiges parametrization are the *Gibbs parameters* [56], defined as

$$\tan(\phi/2)\,\mathbf{a} \tag{2.4}$$

The Gibbs parameters are unique in the sense that each set of parameters in $\mathbb{R}^3$ represents a different rotation. Also, there is a simple formula to compose rotations in Gibbs form. However, the Gibbs form has the disadvantage that rotations around $\pi$ cannot be represented, since $\tan(\pi/2)$ is undefined.

**Rotation Matrix** As each 3D rotations is a linear operation on $\mathbb{R}^3$, it can be represented as $3 \times 3$ matrix $R$, the *rotation matrix*. Its columns (or rows, if multiplied from the right) can be interpreted as the images of the unit vectors under the rotation. Rotation matrices are orthogonal ($R^T R = I$) and have a determinant of 1.

The rotation axis is an Eigenvector for the Eigenvalue 1 of the rotation matrix, while the rotation angle can be computed using acos and the trace *tr* of the matrix

$$tr(R) = 1 + 2\cos(\phi) \tag{2.5}$$

Due to numerical issues with acos, it is more robust to rely on the atan2 function, using the identity

$$2\sin(\phi) = \begin{vmatrix} R_{3,2} - R_{2,3} \\ R_{1,3} - R_{3,1} \\ R_{1,2} - R_{2,1} \end{vmatrix}, \tag{2.6}$$

to compute the angle as

$$\measuredangle R = \measuredangle \begin{pmatrix} R_{1,1} & R_{1,2} & R_{1,3} \\ R_{2,1} & R_{2,2} & R_{2,3} \\ R_{3,1} & R_{3,2} & R_{3,3} \end{pmatrix} = \text{atan2}\left( \begin{vmatrix} R_{3,2} - R_{2,3} \\ R_{1,3} - R_{3,1} \\ R_{1,2} - R_{2,1} \end{vmatrix}, tr(R) - 1 \right) \tag{2.7}$$

Note that some implementations reverse the order of arguments of atan2. Both formulas can be derived from (2.2). The arguments of atan2 reduce to $2\sin(\phi)$ and $2\cos(\phi)$, respectively.

Composing two rotations is the matrix-matrix product of the two rotations, rotating a vector is done by multiplying the vector with the matrix. Inverting the rotation is performed by transposing the matrix, as it is orthogonal.

**Quaternions** While the rotation matrix is very generic and allows many operations to be performed with acceptable performance, its 9 parameters are significantly more than the theoretically required 3 parameters. As a good compromise between usability, performance and storage costs, unit quaternions are often used for representing rotations. Using Quaternions require less computations for most operations, while the downsides – such as have two possible representations for each rotation – are typically manageable.

### 2.1.2 Comparing 3D Rotations

Different metrics were proposed in the literature for comparing 3D rotations, many of which are essentially equivalent but have different computational costs [71]. When necessary in this work, we compare two 3D rotations $R_1, R_2 \in \mathrm{SO}(3)$ using the angle required to rotate one onto the other:

$$d(R_1, R_2) = \measuredangle(R_1^{-1} R_2) \tag{2.8}$$

This distance is the maximal angle between the two rotated variants of a vector $\mathbf{v}$,

$$d(R_1, R_2) = \max_{\mathbf{v} \in R^3} \measuredangle(R_1\mathbf{v}, R_2\mathbf{v}) \tag{2.9}$$

The metric thus has a physical interpretation and is – contrary to many other proposed metrics – independent from the particular representation of the rotations.

### 2.1.3 Finding a Rotation

We will later require a method that, given two vectors $\mathbf{x}$ and $\mathbf{y}$ of equal length, returns a rotation $R$ such that $R\mathbf{x} = \mathbf{y}$. In particular, we are interested in a rotation with an angle of $\pi$, such that $R^{-1} = R$. The problem is thus reduced to finding the corresponding rotation axis $\mathbf{a}$. Formally, we need our axis $\mathbf{a}$ to fulfill

$$R(\mathbf{a}, \pi)\mathbf{x} = \mathbf{y} \tag{2.10}$$

If $\mathbf{x} + \mathbf{y} \neq 0$, this condition is met by

$$\mathbf{a} = (\mathbf{x} + \mathbf{y})/|\mathbf{x} + \mathbf{y}| \tag{2.11}$$

This is obvious from geometric intuition, or by substituting the corresponding values in the Rodruiges' formula (2.1)

$$\begin{aligned}
R(\mathbf{a}, \pi)\mathbf{x} &= \overbrace{\cos\pi}^{=-1}\mathbf{x} + \overbrace{\sin\pi}^{=0}(\mathbf{a} \times \mathbf{x}) + \overbrace{(1 - \cos\pi)}^{=2}\mathbf{a}(\mathbf{a} \cdot \mathbf{x}) \\
&= -\mathbf{x} + 2\mathbf{a}(\mathbf{a} \cdot \mathbf{x}) \\
&= -\mathbf{x} + \frac{2}{|\mathbf{x} + \mathbf{y}|^2}(\mathbf{x} + \mathbf{y})((\mathbf{x} + \mathbf{y}) \cdot \mathbf{x}) \\
&= -\mathbf{x} + \frac{2}{\mathbf{x} \cdot \mathbf{x} + 2\mathbf{x} \cdot \mathbf{y} + \mathbf{y} \cdot \mathbf{y}}(\mathbf{x} + \mathbf{y})(\mathbf{x} \cdot \mathbf{x} + \mathbf{x} \cdot \mathbf{y})
\end{aligned}$$

Since $\mathbf{x}$ and $\mathbf{y}$ must have equal length, $|\mathbf{x}| = |\mathbf{y}| \Leftrightarrow \mathbf{x} \cdot \mathbf{x} = \mathbf{y} \cdot \mathbf{y}$, and we obtain

$$\begin{aligned}
&= -\mathbf{x} + \frac{2}{2\mathbf{x} \cdot \mathbf{x} + 2\mathbf{x} \cdot \mathbf{y}}(\mathbf{x} + \mathbf{y})(\mathbf{x} \cdot \mathbf{x} + \mathbf{x} \cdot \mathbf{y}) \\
&= -\mathbf{x} + (\mathbf{x} + \mathbf{y}) \\
&= \mathbf{y}
\end{aligned}$$

However, for the case that $\mathbf{x} + \mathbf{y} \approx 0 \Leftrightarrow \mathbf{x} \approx -\mathbf{y}$, $(\mathbf{x} + \mathbf{y})/|\mathbf{x} + \mathbf{y}|$ becomes numerically unstable to compute. For $\mathbf{x} + \mathbf{y} = 0$, an arbitrary unit vector that is orthogonal to both $\mathbf{x}$ and $\mathbf{y}$ is a valid rotation vector. In fact, any $\mathbf{a}$ that fulfills the following conditions also fulfills (2.10):

$$\mathbf{x} \cdot \mathbf{a} = \mathbf{y} \cdot \mathbf{a} \;\Leftrightarrow\; (\mathbf{x} - \mathbf{y}) \cdot \mathbf{a} = 0 \tag{2.12}$$

$$|\mathbf{a}| = 1 \;\Leftrightarrow\; \mathbf{a} \cdot \mathbf{a} = 1 \tag{2.13}$$

$$\mathbf{a}|\mathbf{x} + \mathbf{y}| = \mathbf{x} + \mathbf{y} \tag{2.14}$$

Condition (2.12) states that the angle between the axis and the two vectors are identical, while condition (2.14) states that $a$ must be parallel to $\mathbf{x}+\mathbf{y}$, if the latter is non-zero.

To show that any $\mathbf{a}$ that fulfills (2.12-2.14) also fulfills (2.10), let's first consider the case that $|\mathbf{x}+\mathbf{y}| \neq 0 \Leftrightarrow \mathbf{x}+\mathbf{y} \neq 0$. In this case, $\mathbf{a} = (\mathbf{x}+\mathbf{y})/|\mathbf{x}+\mathbf{y}|$, and above's proof applies. In case that

$$|\mathbf{x}+\mathbf{y}| = 0 \Leftrightarrow \mathbf{x} = -\mathbf{y}, \tag{2.15}$$

we obtain

$$\begin{aligned} \mathbf{a}\cdot\mathbf{y} \overset{(2.12)}{=} \mathbf{a}\cdot\mathbf{x} \overset{(2.15)}{=} \mathbf{a}\cdot -\mathbf{x} &\Rightarrow 2(\mathbf{a}\cdot\mathbf{x}) = 0 \\ &\Rightarrow 2\mathbf{a}(\mathbf{a}\cdot\mathbf{x}) = 0 \end{aligned}$$

and thus

$$\begin{aligned} R(\mathbf{a},\pi)\mathbf{x} &= \overbrace{\cos\pi}^{=-1}\mathbf{x} + \overbrace{\sin\pi}^{=0}(\mathbf{a}\times\mathbf{x}) + \overbrace{(1-\cos\pi)}^{=2}\mathbf{a}(\mathbf{a}\cdot\mathbf{x}) \\ &= -\mathbf{x} + 2\mathbf{a}(\mathbf{a}\cdot\mathbf{x}) \\ &= -\mathbf{x} = \mathbf{y} \end{aligned} \tag{2.16}$$

In the general case, we thus look for a normalized axis $\mathbf{a}$ that fulfills both (2.12) and (2.14). This can be modeled as linear system

$$\begin{pmatrix} |\mathbf{x}+\mathbf{y}| & & \\ & |\mathbf{x}+\mathbf{y}| & \\ & & |\mathbf{x}+\mathbf{y}| \\ & \mathbf{x}-\mathbf{y} & \end{pmatrix} \mathbf{a} = \begin{pmatrix} \\ \mathbf{x}+\mathbf{y} \\ \\ 0 \end{pmatrix} \tag{2.17}$$

Note that for $\mathbf{x}+\mathbf{y} \neq 0$, the solution to this system is already normalized, since

$$\begin{aligned} |\mathbf{x}+\mathbf{y}|a_1 &= \mathbf{x}_1 + \mathbf{y}_1 \\ |\mathbf{x}+\mathbf{y}|a_2 &= \mathbf{x}_2 + \mathbf{y}_2 \\ |\mathbf{x}+\mathbf{y}|a_3 &= \mathbf{x}_3 + \mathbf{y}_3 \\ \Rightarrow \mathbf{a}_1^2 + \mathbf{a}_2^2 + \mathbf{a}_3^2 &= \frac{(\mathbf{x}_1+\mathbf{y}_1)^2 + (\mathbf{x}_2+\mathbf{y}_2)^2 + (\mathbf{x}_3+\mathbf{y}_3)^2}{|\mathbf{x}+\mathbf{y}|^2} = 1 \end{aligned} \tag{2.18}$$

**Pragmatic Approach** In some practical setups, where speed is crucial, building and solving (2.17) might be too expensive. A more pragmatic approach is to use (2.11) when the angle between $\mathbf{x}$ and $\mathbf{y}$ does not exceed a certain threshold $\epsilon$, such as $\epsilon = 0.1°$. In all other cases, the method described in Sec. 2.2.1 is used to find a vector orthogonal to $\mathbf{x}$, and that vector is used as rotation axis.

## 2.2 Basic Operations in 3D

This sections describes the implementation of several basic operations on 3D vectors and transformations. They all are of little theoretical interest, but require some careful engineering to avoid common pitfalls.

### 2.2.1 Finding an Orthogonal Vector

**The Challenge** When dealing with 3D data, one often needs to define a local coordinate system given only a point and its normal vector. Such systems are necessary, for example, to compute certain local surface features around a point. The problem of defining a local coordinate frame is equivalent to finding a vector that is orthogonal to the normal vector, since the third axis is simply the cross product of the normal vector and the orthogonal vector.

Of course, such a system is not uniquely defined, since it can be rotated around the normal vector. Another requirement might thus be robustness regarding the direction of the normal vector: When varying the normal only a little, one would like the local system to also change only little. Unfortunately, the hairy ball theorem shows that it is not possible to find such an orthogonal vector in a continuous way.[3] Instead, for each mapping from normal vectors to an orthogonal vector, there is at least one direction of the normal vector where the orthogonal vector flips in some non-continuous way.

For some methods, a unique direction is extracted from the surrounding 3D data. For example, given a point on a 3D surface, a PCA analysis of the surrounding points can be used to extract a local coordinate frame. However, such surrounding 3D data is not always available in our use-cases.

**Our Approach** In practice, given a normal vector $\mathbf{n}$, two of the three unit axis vectors are projected onto the plane orthogonal to $n$. The longest projection is normalized and used as orthogonal vector. This approach avoids problems when $n$ is collinear to one of the two unit vectors. Fig. 2.1 illustrates the resulting directions.

[3]The hairy ball theorem states that given a 3D ball with hairs, it is not possible to comb that ball without introducing a twirl. Each point on the 3D ball can be interpreted as normalized 3D vector, and the direction of the 'hair' attached to that point, which is orthogonal to the vector, defines a local coordinate frame. See, for example, [69].

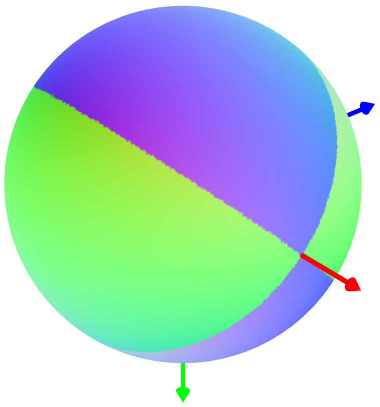


Figure 2.1: Illustration of orthogonal directions, computed using the method described in Sec. 2.2.1. The directions are color coded, using the shown axes. The discontinuous areas are clearly visible, their fuzziness is an artifact of the visualization.

```
Input: Normalized direction $\mathbf{n}$

// Project $e_1$, $e_2$ onto plane defined by $n$
$\mathbf{p}_1 \leftarrow \mathbf{e}_1 - (\mathbf{n} \cdot \mathbf{e}_1)\mathbf{n}$
$\mathbf{p}_2 \leftarrow \mathbf{e}_2 - (\mathbf{n} \cdot \mathbf{e}_2)\mathbf{n}$

// Use larger projection as orthogonal direction, normalize
if $|\mathbf{p}_1| \geq |\mathbf{p}_2|$
    $\mathbf{o} \leftarrow \mathbf{p}_1/|\mathbf{p}_1|$
else
    $\mathbf{o} \leftarrow \mathbf{p}_2/|\mathbf{p}_2|$

Output: Vector $\mathbf{o}$, orthogonal to $\mathbf{n}$
```

### 2.2.2 Finding the Unsigned Angle between Vectors

**acos method** Another common problem is to compute the angle $\measuredangle(\mathbf{a}, \mathbf{b})$ between two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$. A straightforward and common approach is to use the identity of the euclidean dot product,

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos(\measuredangle(\mathbf{a}, \mathbf{b})) \tag{2.19}$$

to compute the angle as

$$\measuredangle(\mathbf{a}, \mathbf{b}) = \cos^{-1}\left(\frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}||\mathbf{b}|}\right). \tag{2.20}$$

However, this equation requires special handling of the case that either of $\mathbf{a}$ or $\mathbf{b}$ is zero, and becomes numerically unstable for $|\mathbf{a}||\mathbf{b}| \to 0$. Also, due to numerical

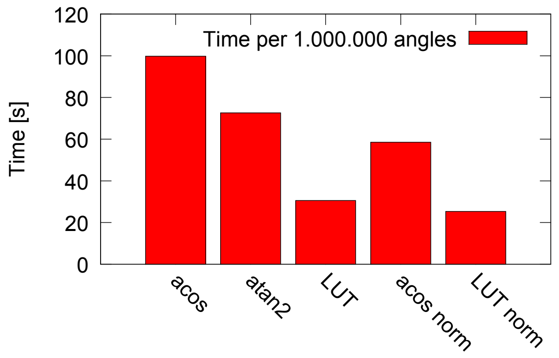


Figure 2.2: Speed comparison of different methods for computing the angle between two 3D vectors. The first three methods show timings for the angle between non-normalized vectors. Methods ending with *norm* assume that the two vectors are normalized.

errors, $(\mathbf{a} \cdot \mathbf{b})/(|\mathbf{a}||\mathbf{b}|)$ sometimes happens to be slightly larger than 1 or smaller than $-1$, such that $\cos^{-1}$ fails with an error or returns a non-normalized floating point number [121].

**atan2 method** It is instead of numerical advantage to use the cross product identity

$$|\mathbf{a} \times \mathbf{b}| = |\mathbf{a}||\mathbf{b}| \sin(\measuredangle(\mathbf{a}, \mathbf{b})) \tag{2.21}$$

along with the atan2-function to obtain

$$\measuredangle(\mathbf{a}, \mathbf{b}) = \operatorname{atan2}(\mathbf{a} \cdot \mathbf{b}, |\mathbf{a} \times \mathbf{b}|) \tag{2.22}$$

Note that even though atan2 returns an angle in $[0, 2\pi[$, [4] (2.22) returns a range of $[0, \pi]$ since $|\mathbf{a} \times \mathbf{b}|$ is non-negative. This is in accordance with the fact that in a 3D space, one cannot define the angle between two vectors such that the full range $[0, 2\pi[$ is used without either making $\measuredangle$ non-commutative or non-rotation-invariant.

While (2.22) is computationally more expensive than (2.20), it is robust for all arguments, numerically more stable and up to 8 orders of magnitude more accurate [121].

**Lookup method** While having a high accuracy, both of the presented methods have the disadvantage of using floating point operations ($\cos^{-1}$, atan2), which can be rather expensive to compute. In some situations – especially in the object

[4] Note that the while the range always has length $2\pi$, its start and end point are implementation and language specific – some systems return the angle in $[-\pi, \pi[$. Note also that for some implementations and languages, the arguments of $atan2$ are reversed. Finally, note that some implementations raise an error in case of $\operatorname{atan2}(0, 0)$; however, our implementation was done in C, where $\operatorname{atan2}(0, 0) = 0$.

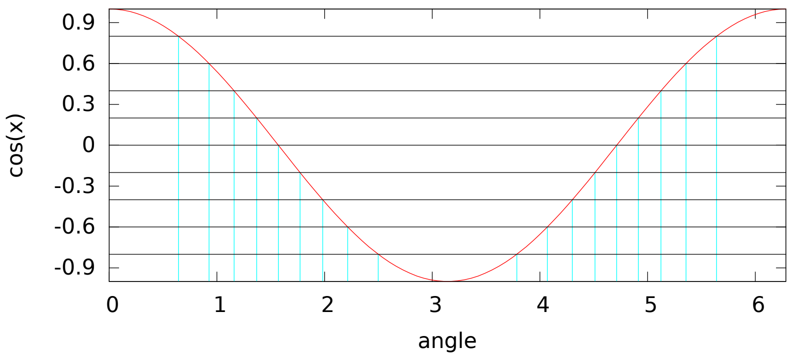


Figure 2.3: Illustration of the accuracy of a lookup table for $cos^{-1}$. A regular sampling of $[-1, 1]$ (here in steps of $s = 0.2$) leads to an irregular sampling of the angular space. Accuracy is worst at the angles $k\pi$, where the derivative of $cos$ is close to zero.

detection scheme presented in Sec. 5 –, if the required accuracy is small and both vectors are normalized, a lookup table can be faster than above's method.

For normalized vectors, we use a lookup table based on

$$d = \mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos(\measuredangle(\mathbf{a}, \mathbf{b})) = \cos(\measuredangle(\mathbf{a}, \mathbf{b})) \tag{2.23}$$

The accuracy is mostly problematic around $\measuredangle(\mathbf{a}, \mathbf{b}) = k\pi$, where the derivative of $cos$ is close to zero. Fig. 2.3 illustrates this and shows that the worst accuracy can be expected around the zero angle. When sampling the interval $[-1, 1]$ in steps of $s$, the accuracy can be estimated as

$$\text{acc} \approx \cos^{-1}(1 - s) \tag{2.24}$$

For a target accuracy of $\text{acc} = \delta$, we can thus estimate the required sampling step size as

$$s \leq 1 - \cos(\delta) \tag{2.25}$$

Fig. 2.3 shows the required step size and number of steps for several target accuracy. Lookup tables beyond an accuracy of around 1° are questionable, since a large number of steps leads to a worsened cache behavior.

Lookup tables provide an elegant way of avoiding the problem that even for normalized vectors, $|\mathbf{a} \cdot \mathbf{b}|$ might be slightly larger than one. By adding additional bins beyond the theoretical range $[-1, 1]$, those cases can be caught without additional costs in terms of range checks.

**Integrated Lookup Table** Note that the use of a lookup table requires a cast from a floating point number to an integer. For some systems, such an operation can be quite time consuming. However, in our usage scenarios, we need to convert the resulting angle into an integer anyway, since we sample the resulting angles

Table 2.3: Required sampling step size and number of steps for the $\cos^{-1}$ lookup table, given some target accuracy $\delta$.

| Target Accuracy $\delta$ in ° | 15 | 10 | 5 | 2 | 1 | 0.1 |
|---|---|---|---|---|---|---|
| Sampling size $s$ | 0.034 | 0.015 | 0.0038 | 0.00061 | 0.00015 | 1.5e-006 |
| Number of steps $\lceil 2/s \rceil$ | 59 | 132 | 526 | 3 284 | 13 132 | 1 313 123 |

with uniform intervals (compare Sec. 5, (5.21) and (5.7)). This conversion can be integrated into the lookup table, such that no additional conversion from float to integers is required.

**Taylor expansion** A final approximate approach we evaluated is to use the taylor expansion of *acos* to speed up the potentially expensive computation of *acos*

$$\operatorname{acos}(x) \approx \frac{\pi}{2} - x - \frac{x^3}{6} - \frac{3x^5}{40} - \frac{5x^7}{112} - \frac{35x^9}{1152} + \mathcal{O}(x^{11}) \tag{2.26}$$

However, we found that even when using the evaluation up to order $x^9$, the error made in the evaluation was up to 15°. When using more terms, the expansion becomes more expensive to compute than when using one of the exact methods. The taylor expansion is therefore either less accurate than the Lookup-Table method, or slower than the exact methods, such that there is no benefit to it.

**Evaluation** Both the acos and the LUT method require a division if the two vectors are non-normalized. In both cases, if the vectors are known to have length 1, the divisions and length computations can be omitted, leading to additional speedup. Fig. 2.2 shows timings for the different proposed methods.

### 2.2.3 Finding the Signed Angle between Vectors

A slightly different problem is to compute the *signed angle* $\measuredangle_d(\mathbf{a}, \mathbf{b})$ between $\mathbf{a}$ and $\mathbf{b}$, given an arbitrary axis $\mathbf{d}$, $|\mathbf{d}| = 1$. It is defined such that

$$R_{\mathbf{d}}(\measuredangle_d(\mathbf{a}, \mathbf{b}))\mathbf{a} = \mathbf{b} \tag{2.27}$$

meaning that if we rotate around $\mathbf{d}$ with an angle of $\measuredangle_d$, $\mathbf{a}$ is mapped onto $\mathbf{b}$. $\measuredangle_d$ has, contrary to $\measuredangle(\mathbf{a}, \mathbf{b})$, the full range $[0, 2\pi[$. Note that $\measuredangle_d(\mathbf{a}, \mathbf{b})$ is defined if and only if

$$\mathbf{a} \cdot \mathbf{d} = \mathbf{b} \cdot \mathbf{d} \tag{2.28}$$

which is easily shown by multiplying (2.1) with $\mathbf{d}$. The following method assumes that the problem is well-formed.

To compute $\measuredangle_d(\mathbf{a}, \mathbf{b})$, we first simplify the problem and assume that $\mathbf{d}$ is orthogonal to both $\mathbf{a}$ and $\mathbf{b}$. In this case, we can use $\mathbf{a} \times \mathbf{b}$ to compare the orientation of $\mathbf{a}$ and $\mathbf{b}$ to $\mathbf{d}$, using

$$s = sgn((\mathbf{a} \times \mathbf{b}) \cdot \mathbf{d}) \tag{2.29}$$

Then,

$$\measuredangle_d(\mathbf{a}, \mathbf{b}) = \begin{cases} \measuredangle(\mathbf{a}, \mathbf{b}) & s \geq 0 \\ 2\pi - \measuredangle(\mathbf{a}, \mathbf{b}) & s < 0 \end{cases} \tag{2.30}$$

The case that $\mathbf{d}$ is not orthogonal to $\mathbf{a}$ and $\mathbf{b}$ can be reduced to above's case by projecting $\mathbf{a}$ and $\mathbf{b}$ onto the plane defined by $\mathbf{d}$:

$$\measuredangle_d(\mathbf{a}, \mathbf{b}) = \measuredangle_d(\mathbf{a} - \mathbf{d}(\mathbf{a} \cdot \mathbf{d}), \mathbf{b} - \mathbf{d}(\mathbf{b} \cdot \mathbf{d})). \tag{2.31}$$

### 2.2.4 Comparing Rigid Transformations

When clustering rigid transformations or when comparing a recovered rigid transformation with a ground truth transformation, one might need some measure to compare two such transformations. Defining a single-valued metric is not trivial, since it requires the combination of rotation and translation, which live in different spaces with different metrics. For example, is a transformation where translation is off by 1 mm better than a transformation where rotation is off by $1°$?

Given two rigid transformations $T_1, T_2 \in \mathrm{SE}(3)$, we first define the difference transformation as

$$D(T_1, T_2) = T_1^{-1} T_2 = (R, \mathbf{t}) \tag{2.32}$$

We use $D$ as basis for any metric, since it is independent of rigid motions: given any motion $M \in \mathrm{SE}(3)$ that is applied to both $T_1$ and $T_2$, we have

$$D(MT_1, MT_2) = (MT_1)^{-1}(MT_2) = T_1^{-1} M^{-1} M T_2 = T_1^{-1} T_2 = D(T_1, T_2) \tag{2.33}$$

**Two-valued metrics** Several strategies exist for defining comparison metrics, some of which are domain-dependent. A very simple one is to ignore the problem of combining translation and rotation and to simply use two measures, typically $|\mathbf{t}|$ for translation and $\measuredangle(R)$ for rotation (see Sec. 2.1.2):

$$m_2(T_1, T_2) = (\measuredangle(R), |\mathbf{t}|) \tag{2.34}$$

If the transformation is to be applied to some object with a fixed, known size (for example, when localizing a chair for which a CAD model is available), the size of the object can be used to normalize the translational part of $m_2$. The normalized metric is

$$m_{2,\text{norm}} = \left( \measuredangle(R), \frac{|\mathbf{t}|}{\text{diam}(M)} \right). \tag{2.35}$$

This allows to define absolute, object-independent thresholds when comparing a rigid transformation with some ground truth transformation. For example, one can define a recovered transformation to be correct if $m_{2,\text{norm}} < (10°, 0.1)$, meaning that the translation may not be off by more than 10% of the object's diameter, and the rotation not more than 10°.

**Single-valued metrics** If again a model $M$ of the transformed object is available, one can define a single-valued metric for comparing two transformations. For this, we apply both transformations onto $M$ and measure by how much the points differ after the transformation. Formally, we define

$$\begin{aligned} m_1 &= \max\{|T_1\mathbf{x} - T_2\mathbf{x}| : \mathbf{x} \in M\} \\ &= \max\{|\mathbf{x} - D\mathbf{x}| : \mathbf{x} \in M\}. \end{aligned} \tag{2.36}$$

We can again define a normalized variant of this metric as

$$m_{1,\text{norm}} = \frac{m_1}{\text{diam}(M)} \tag{2.37}$$

The metric $m_{1,\text{norm}}$ has the advantage that it requires only a single threshold for comparing rigid transformations.

**Efficient Computation** For large models with many points, $m_1$ might be expensive to compute. However, one can speed up the computation using the observation that $m_1$ needs to be evaluated only for the convex hull $C(M)$ of $M$: For $\mathbf{x} \in M$, $\mathbf{x} = \sum_i \alpha_i \mathbf{c}_i$ with $\mathbf{c}_i \in C(M)$, $\sum_i \alpha_i = 1$, $0 \leq \alpha_i \leq 1$, we get

$$\begin{aligned} |\mathbf{x} - D\mathbf{x}| &= \left| \sum_i \alpha_i \mathbf{c}_i - D \sum_i \alpha_i \mathbf{c}_i \right| = \left| \sum_i \alpha_i \mathbf{c}_i - \sum_i \alpha_i D\mathbf{c}_i \right| = \left| \sum_i \alpha_i (\mathbf{c}_i - D\mathbf{c}_i) \right| \\ &\leq \sum_i \alpha_i |\mathbf{c}_i - D\mathbf{c}_i| \leq \max_i |\mathbf{c}_i - D\mathbf{c}_i| \end{aligned} \tag{2.38}$$

With the same reasoning, one can compute an approximating upper bound of $m_1$ by using the 8 corners of a bounding box of $M$. We call call this approximation $m_{1,approx}$, and the corresponding normalized metric $m_{1,approx,\text{norm}}$.

## 2.3 Sampling

**Motivation** Uniform sampling of 3D point clouds serves two main purposes: It eliminates bias towards scene parts that are more densely sampled, and it improves performance by reducing the cloud to only as few points as necessary.

In scenes captured with a depth sensor that follows a pinhole-based camera model, such as time-of-flight or most stereo setups, scene parts closer to the camera center will have more points per unit surface that those further away. Also, noise, occlusion and other acquisition-specific characteristics can lead to irregular densities throughout the scene. The proposed detection scheme uses a voting approach, where scores are computed for pose hypotheses based on the number of scene points that would lie on the object given the pose candidate. The voting would thus be sensitive to the non-uniform sampling mentioned before, introducing a bias to scene parts that are more densely sampled.

Additionally, the proposed detection scheme is able to to find pose candidates with as few as a 100 points on the target object. Since high-resolution sensors can easily produce point clouds containing some million points, reducing the number of points can lead to a just as accurate, but faster detection.

**Method** A *voxel based sampling approach* would overlay the scene with a voxel grid and keep at most a single point per voxel. While fast, this approach has the disadvantage that it is not rotation invariant and that points in neighboring voxels might still be close to each other. Instead, we use a *greedy sampling approach* that iterates over the points of the original point cloud and adds them to the sampled point cloud only if there is no other point in the sampled cloud that is closer to the new point than the given sampling distance. In pseudo code, this reads

```
Input: Input point cloud $C_{in}$
       Sampling distance $d_{max}$

$C_{out} \leftarrow \emptyset$
for each $v \in C_{in}$
  if $\min_{w \in C_{out}} |v - w| > d_{max}$
    $C_{out} \leftarrow C_{out} \cup \{v\}$

Output: Sampled point cloud $C_{out}$
```

A spatial index is used to speedup the search for the closest point in the already sampled cloud. Even though this approach is not straightforward to parallelize, it is invariant to rigid transformations, obeys the distance criterion strictly, and is easily extendable to take normal directions into account.

# Part II

# Nearest Neighbors and Pose Refinement in 3D

# 3 Voxel-Hash Based Nearest Neighbor Search

A recurring problem in 3D applications are 3D nearest neighbor lookups. For example, most of the runtime of the iterative closest point (ICP) algorithm is traditionally spent in the nearest neighbor lookup. Another application are surface comparisons, where the point-to-point distances between two registered point clouds need to be computed.

In this chapter, a novel method for exact and approximate 3D nearest neighbor lookups is proposed that allows almost constant-time lookups. Most notably and contrary to previous approaches, lookup times are nearly independent of the distribution of data and query points, allowing usage of the method in real-time scenarios. The lookup times of the proposed method outperform prior art sometimes by several orders of magnitude. This speedup is bought at the price of increased costs for creating the indexing structure. However, this can typically be done in the offline phase. Additionally, an approximate variant of the method is proposed that significantly reduces the time required for data structure creation.

Parts of this chapter previously appeared in [42].

## 3.1 Introduction

Quickly finding the closest point from a large set of data points in 3D is crucial for alignment algorithms, such as ICP, as well as industrial inspection and robotic navigation tasks. Most state-of-the-art methods for solving the nearest neighbor problem in 3D are based on recursive subdivisions of the underlying space to form a tree of volumes. The various subdivision strategies include uniform subdivisions, such as octrees [90], as well as nonuniform subdivisions, such as $k$-d-trees [13] and Delaunay- or Voronoi-based subdivisions [39].

Tree-based methods require two steps to find the exact nearest neighbor. First, the query point descends the tree to find its corresponding leaf node. Since the query point might be closer to the boundary of the node's volume than to the data points contained in the leaf node, tree backtracking is required as a second step to search neighboring volumes for the closest data point. The proposed method improves the time for finding the leaf node and removes the need for potentially expensive backtracking by using voxels to recursively subdivide space. The leaf voxel that contains the query point is found by bisecting the voxel level. For trees of depth $L$, this approach requires only $\mathcal{O}(\log(L))$ operations, instead of $\mathcal{O}(L)$ operations when letting the query point descend the tree. In addition, each voxel contains a list of all data points whose Voronoi cells intersect that voxel, such that no backtracking is necessary. By storing the voxels in a hash table and enforcing a limit on the number of Voronoi intersections per voxel, the total query time is independent of the position of the query point and the distribution of data points. The theoretical query time is of magnitude $\mathcal{O}(\log(\log(N))$, where $N$ is the size of the target data point set.

The amount of backtracking that is required in tree-based methods depends on the position of the query point. Methods based on backtracking therefore have non-constant query times even when using the same dataset, making them difficult to use in real-time applications. Since the proposed method does not require backtracking, the query time becomes almost independent of the position of the query point. Further, the method is largely parameter free, does not require an a-priori definition of a maximum query range, and is straightforward and easy to implement.

We evaluate the proposed method on different synthetic datasets that show different distributions of the data and query point sets, and compare it to several state of the art methods: a self-implemented $k$-d-tree, the *Approximate Nearest neighbor* (ANN) library [98] (which, contrary to its name, allows also to search for exact nearest neighbors), and the *Fast Library for Approximate Nearest Neighbors* (FLANN) [99]. The experiments show that the proposed method is significantly faster for larger data sets and shows an improved asymptotic behavior. As a trade-off, the proposed method uses a more expensive preprocessing step. We also evaluate an extension of the method that performs approximate nearest neighbor lookups, which is faster for both the preprocessing and the lookup steps. Finally, we demonstrate the performance of the proposed method within two applications on real-world datasets, pose refinement and surface inspection. The runtime of both applications is dominated by the nearest neighbor lookups, which is why both greatly benefit from the proposed method.

## 3.2 Related Work

An extensive overview over different nearest neighbor search strategies can be found in [120]. Nearest-neighbor search strategies can roughly be divided into tree-based and hash-based approaches. Concerning tree-based methods, variants of the *k*-d-tree [13] are state-of-the-art for applications such as ICP, navigation and surface inspection [50]. For high-dimensional datasets, such as images or image descriptors, embeddings into lower-dimensional spaces are sometimes used to reduce the complexity of the problem [72].

Many methods were proposed for improving the nearest neighbor query time by allowing small errors in the computed closest point, i.e., by solving the approximate nearest neighbor problem [4, 61, 30]. While faster, using approximations changes the nature of the lookup and is only applicable for methods such as ICP, where a small number of incorrect correspondences can be dealt with statistically. The iterative nature of ICP can be used to accelerate subsequent nearest neighbor lookups through caching [103, 60]. Such approaches are, however, only usable for ICP and not for defect detection or other tasks.

Yan and Bowyer [155] proposed a regular 3D grid of voxels that allow constant-time lookup for a closest point, by storing a single closest point per voxel. However, such fixed-size voxel grids use excessive amounts of memory and require a tradeoff between memory consumption and lookup speed. The proposed multi-level adaptive voxel grid overcomes this problem, since more and smaller voxels are created only at the interesting parts of the data point cloud, while the speed advantage of hashing is mostly preserved. Glassner [57, 34] proposed to use a hash-table for accessing octrees, which is the basis for the proposed approach.

Using Voronoi cells is a natural way to approach the nearest neighbor problem, since a query point is always contained in the Voronoi cell of its nearest neighbor. Boada *et al.* [20] proposed an octree that approximates generalized Voronoi cells and that can be used to approximately solve the nearest neighbor problem [19]. Their work also gives insight into the construction costs of such an octree. Contrary to the proposed algorithm, their work concentrates on the construction of the data structure and solves the nearest neighbor problem only approximately. Additionally, their proposed octree still requires $\mathcal{O}(depth)$ operations for a query. However, their work indicates how the proposed method can be generalized to other metrics and to shapes other than points. Similar, Har-Peled [63] proposed an octree-like approximation of the Voronoi tesselation. Birn *et al.* [18] proposed a full hierarchy of Delaunay triangulations for 2D nearest neighbor lookups. However, the authors state that their approach is unlikely to work well in 3D and beyond.

## 3.3 Method

### 3.3.1 Notation and Overview

We denote points from the original data set as $\mathbf{x} \in D$ and points of the query set $\mathbf{q} \in Q$. Given a query point $\mathbf{q}$, the objective is to find a closest point $\mathrm{NN}(\mathbf{q}, D) =$

Table 3.1: Notations used to describe the voxel-based nearest neighbor search.

| Notation | Description |
|---|---|
| $D \subset \mathbb{R}^3$ | Target point cloud |
| $N = \|D\|$ | Size of target point cloud |
| $\mathrm{NN}(\mathbf{q}, D)$ | Nearest neighbor of $\mathbf{q} \in \mathbb{R}^3$ in $D$ |
| $\mathrm{ANN}(\mathbf{q}, D)$ | Approximate nearest neighbor |
| $\mathrm{voro}(\mathbf{x})$ | Voronoi cell of $\mathbf{x}$ in $D$: $\mathrm{voro}(\mathbf{x}) = \{\mathbf{y} \in \mathbb{R}^3 : \forall \hat{\mathbf{x}} \in D : \|\mathbf{y} - \mathbf{x}\| \leq \|\mathbf{y} - \hat{\mathbf{x}}\|\}$. |
| $v \subset \mathbb{R}^3$ | A voxel in 3D space |
| $l(v)$ | Voxel level |
| $L(D, v)$ | Set of all data points whose Voronoi cell intersects with voxel $v$ |
| $M_{\max}$ | Maximum list length |
| $L_{\max}$ | Maximum voxel level |

$\arg\min_{\mathbf{x} \in D} |\mathbf{q} - \mathbf{x}|_2$. The individual Voronoi cells of the Voronoi diagram of $D$ are denoted $\mathrm{voro}(\mathbf{x})$, which we see as closed set. Fig. 3.1 summarizes the notations use throughout this chapter.

Note that the nearest neighbor of $\mathbf{q}$ in $D$ is not necessarily unique, since multiple points in $D$ can have the same distance to $\mathbf{q}$. In the practical applications of this method, however, we are mostly interested in a single nearest neighbor. Additionally, considering rounding errors and floating point accuracy, it is highly unlikely for a measured point to actually have multiple nearest neighbors in practice. We will therefore talk of *the* nearest neighbor throughout this chapter, even though this is technically incorrect.

The proposed method requires a pre-processing step where the voxel hash structure for the data set $D$ is created. Once this data structure is precomputed, it remains unchanged and can be used for subsequent queries. The creation of the data structure is done in three steps: The computation of the Voronoi cells for the data set $D$, the creation of the octree and the transformation of the octree into a hash table.

### 3.3.2 Octree Creation

Using Voronoi cells is a natural way to approach the nearest neighbor problem. A query point $\mathbf{q}$ is always contained within the Voronoi cell of its closest point, i.e., $\mathbf{q} \in \mathrm{voro}(\mathrm{NN}(\mathbf{q}, D))$. Thus, finding a Voronoi cell that contains $\mathbf{q}$ is equivalent to finding $\mathrm{NN}(\mathbf{q}, D)$. However, the irregular and data-dependent structure of the Voronoi tessellation does not allow a direct lookup. We thus use the octree to create a more regular structure on top of the Voronoi diagram, which allows to find the corresponding Voronoi cell quickly.

After computing the Voronoi cells for the data set $D$, an octree is created,

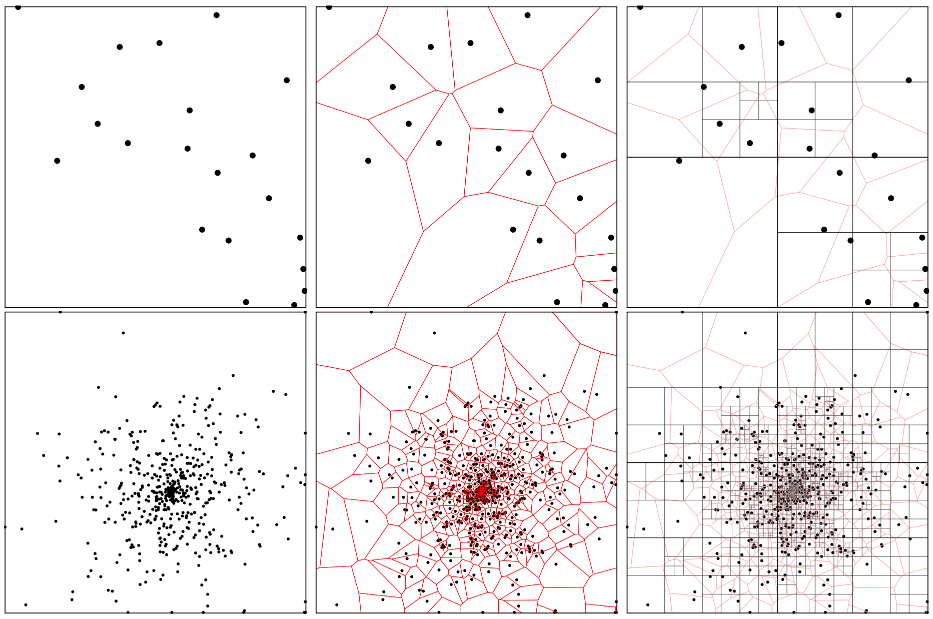


Figure 3.1: Toy example in 2D of the creation of the hierarchical voxel structure. For the data point set (left), the Voronoi cells are computed (center). Starting with the root voxel that encloses all points, voxels are recursively split if the number of intersecting Voronoi cells exceeds $M_{\max}$. In this example, the root voxel is split until each voxel intersects at most $M_{\max} = 5$ Voronoi cells (right).

whose root voxel contains the expected query range. Note that the root voxel can be several thousand times larger than the extend of the data set without significant performance implications.

Contrary to traditional octrees, where voxels are split based on the number of contained data points, we split each voxel based on the number of intersecting Voronoi cells: Each voxel that intersects more than $M_{\max}$ Voronoi cells is split into eight sub-voxels, which are processed recursively. Fig. 3.1 shows a 2D example of this splitting. The set of data points whose Voronoi cells intersect a voxel $v$ is denoted

$$L(D, v) = \{\mathbf{x} \in D : \mathrm{voro}(\mathbf{x}) \cap v \neq \emptyset\}. \tag{3.1}$$

This splitting criterion allows a constant processing time during the query phase: For any query point $\mathbf{q}$ contained in a leaf voxel $v_{\text{leaf}}$, the Voronoi cell of the closest point $\mathrm{NN}(\mathbf{q}, D)$ must intersect $v_{\text{leaf}}$. Therefore, once the leaf node voxel that contains $\mathbf{q}$ is found, at most $M_{\max}$ data points must be searched for the closest point. The given splitting criterion thus removes the requirement for backtracking.

The cost for this is a deeper tree, since a voxel typically intersects more Voronoi cells than it contains data points. The irregularity of the Voronoi tessellation and possible degenerated cases, as discussed below, make it difficult to give

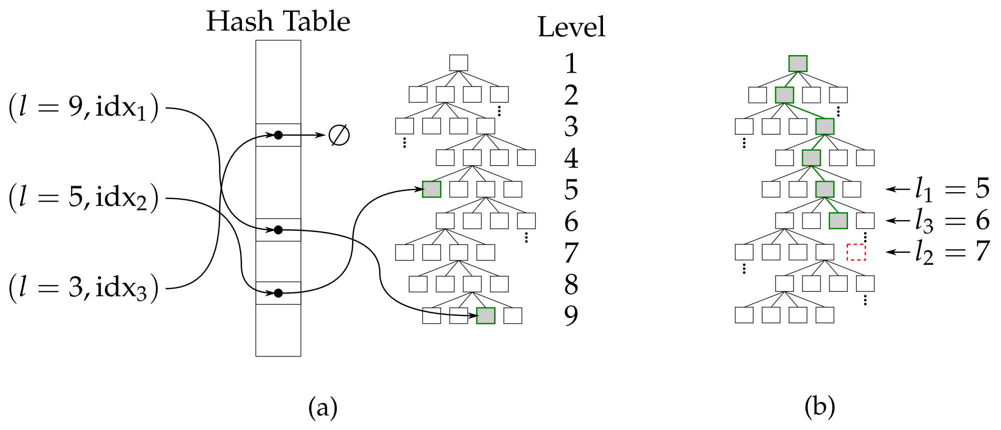

Hash Table
Level
$(l = 9, \mathrm{idx}_1)$
$(l = 5, \mathrm{idx}_2)$
$(l = 3, \mathrm{idx}_3)$
$\emptyset$
1
2
3
4
5
6
7
8
9
$l_1 = 5$
$l_3 = 6$
$l_2 = 7$
(a)
(b)

Figure 3.2: (a) The hash table stores all voxels $v$, which are indexed through their level $l$ and their index idx that contains the integer-valued coordinates of the voxel. The hash table allows to check for the existence of a voxel in constant time. (b) Toy example in 2D of how to find the leaf voxel by bisecting its level. Finding the leaf node by letting the query point descend the tree would require $\mathcal{O}(depth)$ operations on average (green path). Instead, the leaf node is found through bisection of its level. In each step, the hash table is used to check for the presence of the corresponding voxel. The search starts with the center level $l_1 = 5$ and, since the voxel exists, proceeds with $l_2 = 7$. Since the voxel at level $l_2$ does not exist, level $l_3 = 6$ is checked and the leaf node is found.

theoretical bounds on the depth of the octree. However, experimental validation shows that the number of created voxels scales linearly with the number of data points $|D|$ (see Fig. 3.5a).

### 3.3.3 Hash Table

The result of the recursive subdivision is an octree, as depicted in Fig. 3.1. To find the closest point of a given query point $\mathbf{q}$, two steps are required: Find the leaf voxel $v_{\mathrm{leaf}}(\mathbf{q})$ that contains $\mathbf{q}$ and search all points in $L(D, v_{\mathrm{leaf}}(\mathbf{q}))$ for the closest point of $q$. The computation costs for finding the leaf node are on average $\mathcal{O}(depth) \approx \mathcal{O}(\log(|D|))$ when letting $\mathbf{q}$ descend the tree in a conventional way. We propose to use the regularity of the octree to reduce these costs to $\mathcal{O}(\log(depth)) \approx \mathcal{O}(\log(\log(|D|)))$. For this, all voxels of the octree are stored in a hash table that is indexed by the voxel's level $l(v)$ and the voxel's integer-valued coordinates $\mathrm{idx}(v) \in \mathbb{Z}^3$ (Fig. 3.2a).

The leaf voxel $v_{\mathrm{leaf}}(\mathbf{q})$ is then found by bisecting its level. The minimum and maximum voxel level is initialized as $l_{\min} = 1$ and $l_{\max} = depth$. The existence of the voxel with the center level $l_c = \lfloor (l_{\min} + l_{\max})/2 \rfloor$ is tested using the hash table. If the voxel exists, the search proceeds with the interval $[l_c, l_{\max}]$. Otherwise, it proceeds to search the interval $[l_{\min}, l_c - 1]$. The search continues until the interval contains only one level, which is the level of the leaf voxel $v_{\mathrm{leaf}}(\mathbf{q})$. Fig. 3.2 illustrates this bisection on a toy example.

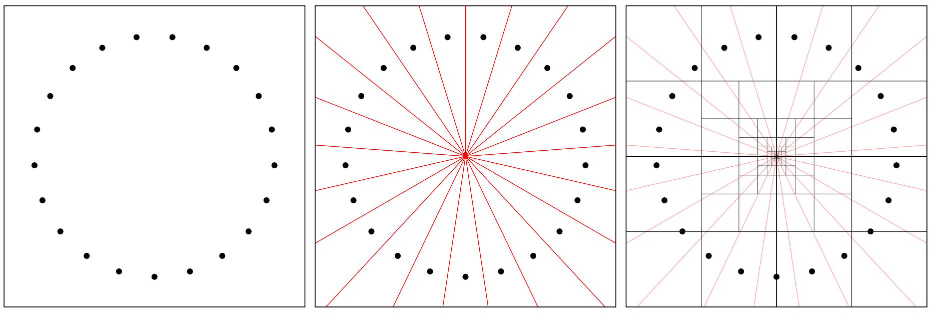


Figure 3.3: Example of a degenerated point set (left) where many Voronoi cells meet at one point (center). In this case, the problem of finding the nearest neighbor is ill-posed for query points close to the center of the circle. To capture such degenerated cases, voxel splitting is stopped after $L_{\max}$ subdivisions (right). See the text for more comments on why such situations are not of practical interest.

Note that in our experiments, tree depths were in the order of 20-40 such that the expected speedup over the traditional method was around 5. Additionally, each voxel in the hash table contains the minimum and maximum depth of its subtree to speedup the bisection. Additionally, the lists $L(D, v)$ are stored only for the leaf nodes. The primary cost during the bisection are cache misses when accessing the hash table. Therefore, an inlined hash table is used to reduce the average amount of cache misses.

### 3.3.4 Degenerated Cases

For some degenerated cases, the proposed method for splitting voxels based on the number of intersecting Voronoi cells might not terminate. This happens when more than $M_{\max}$ Voronoi cells meet at a single point, as depicted in Fig. 3.3. To avoid infinite recursion, a limit $L_{\max}$ on the depth of the octree is enforced. In such cases, the query time for points that are within such an unsplit leaf voxel is larger than for other query points. However, we found that in practice such cases appear only on synthetic datasets. Also, since the corresponding leaf voxels are very small (of size $2^{-L_{\max}} timesthesizeoftherootvoxel)$, chances of a random query point to be within the corresponding voxel are small. Additionally, note the problem of finding the closest point is ill-posed in situations where many Voronoi cells meet at a single point and the query point is close to that point: Small changes in the query point can lead to arbitrary changes of the nearest neighbor. The degradation in query time can be avoided by limiting the length of $L(D, v)$ of the corresponding leaf voxels. The maximum error made in this case is in bound by the diameter of the voxel of level $L_{\max}$. For example, $L_{\max} = 30$ reduces the error to $2^{-30}$ times the size of the root voxel, which is already smaller than the accuracy of single-precision floating point numbers. Summing up, the proposed method degrades only in artificial situations where the problem itself is ill-posed, but the method's performance guarantee can be restored at the cost of an arbitrary small error.

### 3.3.5 Generalizations to Higher Dimensions

The proposed method theoretically can be generalized to dimensions $d > 3$. However, memory and computational costs would likely render the method practically unusable in higher dimensions. This is due to several reasons:

- The branching factor $2^d$ of the corresponding hypercube tree leads to exponentially increasing memory and computation requirements, even for approximately constant average tree depths. For example, even a moderate dimension such as $d = 16$ has a branching factor of $2^{16} = 65536$, such that a tree of depth 3 would already have $(2^{16})^3 = 2^{48}$ nodes.
- Voronoi cells in higher dimensions are increasingly difficult to compute. Dwyer [49] showed that the geometric complexity of the Voronoi cells of $n$ points in dimension $d$ is at least[1]
$$\mathcal{O}(nd^d) \tag{3.2}$$
- Due to the *curse of dimensionality*, the distances between random points in higher dimensions tend to become more similar [12].[2] As one consequence, the number of Voronoi neighbors of each point increase, up to the point where almost all points are neighbors of each other. As another consequence, nearest neighbor lookups for a random query point become ill-conditioned in the sense that a random query point will have many neighbors with approximately equal distance. Voxels are therefore likely to have very long lists of possible nearest neighbors, resulting in deeper voxel trees.

### 3.3.6 Approximate Methods

**Definition** *Approximate nearest neighbor methods* are methods that return only an approximation of the correct nearest neighbor. Approximate methods often are significantly faster or require less memory than exact methods. For example, a simple approximate method is to use a $k$-d-tree without performing backtracking (see, for example, [98, 99]).

Given a query point $\mathbf{q}$ and a dataset $D$, we denote $\text{ANN}(\mathbf{q}, D)$ for an approximate nearest neighbor of $\mathbf{q}$ in $D$. We define the distance to the exact and the

[1]Note that Dwyer showed that for a *fixed* dimension, the complexity is linear in the number of points. Refer to equation (3.3) in [49] and the following discussion for the result regarding (3.2).

[2]The term goes back to Richard E. Bellmann. It captures the fact that even for moderately larger dimensions, the volume of space increases drastically. This often results in counterintuitive effects if one keeps only 3D spaces in mind.

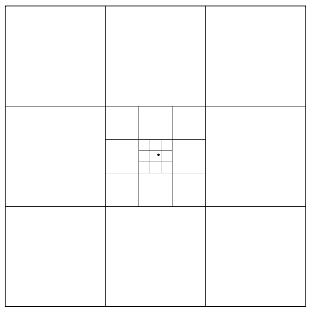


Figure 3.4: 2D-illustration of the fast, approximate voxel creation. Instead of computing and intersecting Voronoi cells, each point (black dot) is added to a $n \times n$-neighborhood (here $3 \times 3$) of voxels, on each level.

approximate nearest neighbor as

$$d_{\mathrm{E}} = |\mathbf{q} - \mathrm{NN}(\mathbf{q}, D)| \tag{3.3}$$

$$d_{\mathrm{A}} = |\mathbf{q} - \mathrm{ANN}(\mathbf{q}, D)| \tag{3.4}$$

with $d_{\mathrm{A}} \geq d_{\mathrm{E}}$.

**Quality Metrics** Several quantitative values can be used to describe the quality of an approximate method. The error probability $p_{\mathrm{err}}$ defines the probability for a random query point to not return the exact, but only an approximate nearest neighbor:

$$p_{\mathrm{err}} = P(d_{\mathrm{A}} > d_{\mathrm{E}}) \tag{3.5}$$

The absolute error is given as

$$E_{\mathrm{abs}} = |d_{\mathrm{A}} - d_{\mathrm{E}}| = d_{\mathrm{A}} - d_{\mathrm{E}} \tag{3.6}$$

Approximate methods are often classified according to the $\epsilon$-criterion, which states that

$$d_{\mathrm{A}} \leq (1 + \epsilon)\, d_{\mathrm{E}} \tag{3.7}$$

and thus puts an upper bound on the relative error.

Given some object $M$ with a fixed, known size $\mathrm{diam}(M)$, we will also measure the quality of an approximate nearest neighbor relative to the object's diameter:

$$E_{rel,M} = E_{\mathrm{abs}} / \mathrm{diam}(M) = (d_{\mathrm{A}} - d_{\mathrm{E}}) / \mathrm{diam}(M) \tag{3.8}$$

The proposed voxel hash method can easily be converted into an approximate method. We will combine two techniques that work at different steps of the method: List length limiting and explicit voxel neighborhood.

**List Length Limiting** A straightforward way of reducing the complexity of both the offline and online phase is to limit the list lengths of each voxel. This is equivalent to storing, for each leaf node, only a subset of the intersecting Voronoi cells. We denote $L_\mathrm{A}$ for a subset of the correct list:

$$L_\mathrm{A}(D,v) \subset L(D,v) \tag{3.9}$$

Several possibilities exist how $L_\mathrm{A}$ can be selected from $L$.

- *Minimize error probability*: Given a voxel $v$, the probability that an intersecting Voronoi cell $\mathrm{voro}(\mathbf{x})$, $\mathbf{x} \in L(D,v)$ contains a query point $\mathbf{q} \in v$ is

  $$P(\mathbf{q} \in \mathrm{voro}(\mathbf{x}) \mid \mathbf{q} \in v) = \frac{\mathrm{vol}(\mathrm{voro}(\mathbf{x}) \cap v)}{\mathrm{vol}(v)} \tag{3.10}$$

  where $\mathrm{vol}(X)$ is the volume of a 3D set $X$.

  Therefore, if $\mathbf{x}$ is removed from $L(D,v)$, the probability of making an approximation error when querying for $\mathbf{q}$ is $P(\mathbf{q} \in \mathrm{voro}(\mathbf{x})|\mathbf{q} \in v)$. In order to minimize the probability of making an error, the points in $L(D,v)$ can be removed based on the volume $\mathrm{vol}(\mathrm{voro}(\mathbf{x}) \cap v)$ of the intersection, removing cells with smaller intersection volumes first. Since the Voronoi cells are disjoint, the total probability of an approximation error is the sum of (3.10) over all removed entries.

  If the approximation error probability shall be bounded, one can remove points from the lists $L(D,v)$ only until said probability is reached.

- *Minimize maximum absolute error*: The Voronoi cells intersecting a voxel can be removed such that some predefined maximum absolute error is maintained. Given some closed, bounded volume $V \subset \mathbb{R}^3$, we define the maximum distance of a point inside that volume from the volume's boundary,

  $$\mathrm{maxdist}(V) = \sup_{\mathbf{v} \in V} \inf_{\mathbf{w} \in \mathbb{R}^3 \setminus V} |\mathbf{v} - \mathbf{w}| \tag{3.11}$$

  If an entry $\mathbf{x} \in L(D,v)$ is removed from $L(D,v)$, the maximum absolute error possible is

  $$\mathrm{maxdist}(\mathrm{voro}(\mathbf{x}) \cap v) \tag{3.12}$$

  If multiple entries $\mathbf{x}_1, \mathbf{x}_2, \ldots$ are removed, the maximum absolute error is

  $$\max E_\mathrm{abs} = \mathrm{maxdist}\left(\bigcup_i (\mathrm{voro}(\mathbf{x}_i) \cap v)\right) \tag{3.13}$$

  This formula allows to remove points from $L(D,v)$ while keeping a bound on the maximum absolute error.

- *Greedy element selection*: Both methods above require an explicit computation of the Voronoi cells and their intersection with voxels. While elegant, such computations can be expensive.

  A different strategy is to keep a fixed number of vertices that are closest to the center of the voxel. This strategy is faster, since it does not require explicit computation of the intersection volumes. It is especially efficient in combination with the next step, which avoids constructing Voronoi cells all together.

**Explicit Voxel Neighborhood** As shown in Sec. 3.4, using Voronoi cells as described leads to a potentially very time-consuming offline stage. Most of the runtime is spent in the creation of the Voronoi cells, and the intersection between Voronoi cells and voxels.

A different approach allows a much faster assignment of points to voxels: Instead of intersecting Voronoi cells with voxels, a point is added to the list of its neighboring voxels only. Fig. 3.4 illustrates this: The given point is added to all voxels in its $3 \times 3$ (or, in 3D, $3 \times 3 \times 3$) neighborhood.

This technique is combined with the list length limiting by retaining only a few or even one point that is closest to the voxel's center. The runtime for creating the voxel tree this way is linear in the number of points $N$ and has a significantly smaller constant factor. In particular, no complex creation of Voronoi cells needs to be performed.

Note that both steps modify only the creation of the data structure; the lookup phase stays the same. The following algorithm summarizes the proposed approximate method.

```
Input: Dataset $D$
       Voxel level range $l_{\min}$ and $l_{\max}$
       Maximum list length $M_{\max}$

for $\mathbf{x}$ in $D$ do:
  for $l$ from $l_{\min}$ to $l_{\max}$ do:
    $v \leftarrow$ voxel of level $l$ containing $\mathbf{x}$
    for $v'$ in $3 \times 3 \times 3$ neighborhood of $v$:
      $L(v') \leftarrow L(v') \cup \{\mathbf{x}\}$

for all voxels $v$:
  Sort $L(v)$ by distance of $\mathbf{x} \in L(v)$ to center of $v$
  Truncate $L(v)$ at length $M_{\max}$

Output: Set of voxel lists $L$
```

**Tree Depth** For the exact methods, voxels were split based on the number of intersecting Voronoi cells. This provided a natural way of splitting voxels only

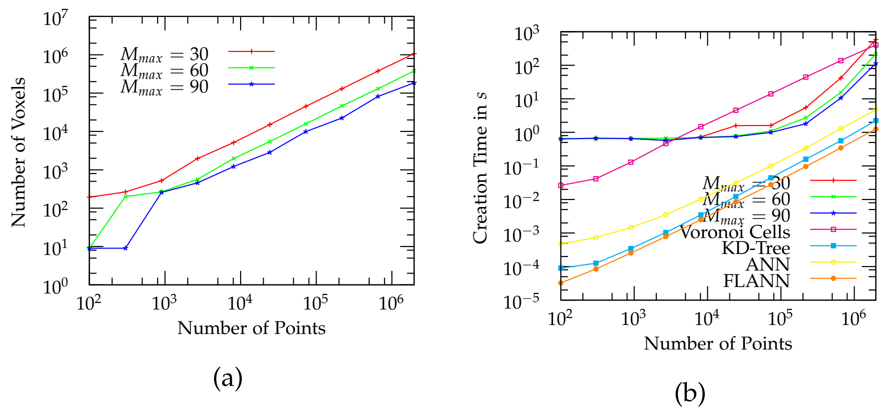


Figure 3.5: Construction and memory costs of the proposed data structure for the CLUSTER dataset. (a) The number of created voxels depends linearly on the size of the data cloud. As a rule of thumb, one voxel is created per data point. (b) The creation time of the voxel data structure. The creation of the Voronoi cells is independent of the value of $M_{\text{max}}$ and its creation time is plotted separately. Although the creation of the voxel data structure is significantly slower than for the $k$-d-tree, the ANN and the FLANN libraries, the creation times are still reasonable for offline processing. Note that the constant performance of the proposed method for less than $10^5$ data points is based on our particular implementation, which is optimized for large data sets and requires constant time for the creation of several caches. Overall, larger values of $M_{\text{max}}$ lead to faster and less memory consuming data structure creation, at the expense of matching time (see Fig. 3.8).

where necessary. A downside of the proposed approximate method is that this automatic splitting no longer happens. As consequence, the range of levels must be specified a priori.

Two use-cases of the proposed method are nearest neighbor lookups for ICP and for defect detection. For both, the set of data points is regularly sampled (see Sec. 5) with some distance $d_{\text{sampling}}$. This allows to simply use $d_{\text{sampling}}$ as a lower bound on the voxel size.

Additionally, a post-processing step can be used to remove unnecessary voxels: If only a single point is stored for each voxel ($M_{\text{max}} = 1$), and all existing child voxels of some voxel $v$ store the same point, then all those child voxels can be removed without changing the result of the nearest neighbor lookup. This effectively prunes the voxel tree at uninteresting locations.

## 3.4 Experiments

Several experiments were conducted to evaluate the performance of the proposed method in different situations and to compare it to the $k$-d-tree, the ANN

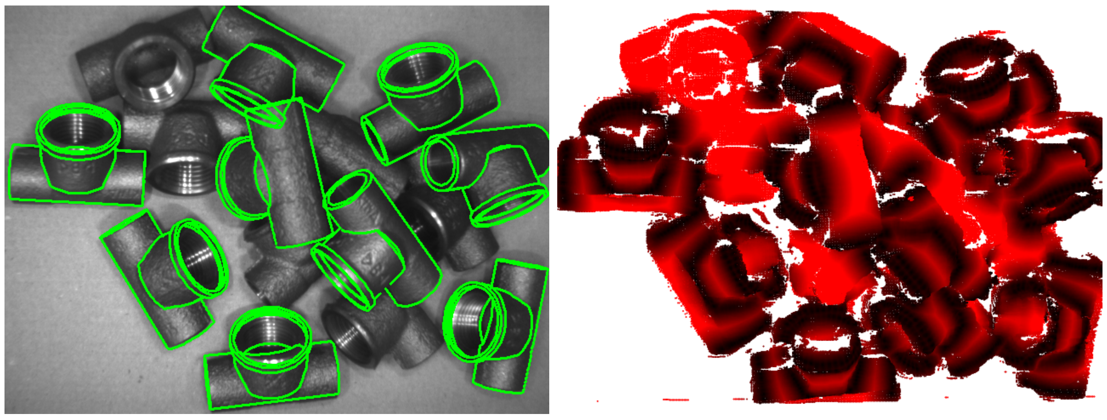


Figure 3.6: Example application for the proposed method. A 3D scan of the scene was acquired using a multi-camera stereo setup and approximate poses of the pipe joint were found using the method of Sec. 5. (a) The poses were refined using ICP of Sec. 4. The corresponding nearest neighbor lookups were logged and used for the evaluation shown in Tab. 3.2. (b) For each scene point close to one of the detected objects, the distance to the object is computed and visualized. This allows the detection of defects on the surface of the objects. The lookups were again logged and used for the performance evaluation in Table 3.2.

libary [98] and the FLANN library [99] as state-of-the-art methods. Note that the FLANN library returns an approximate nearest neighbor, while ANN was configured such that an exact nearest neighbor was returned. Both the *k*-d-tree and the voxel hash structure were implemented in C with similar optimization. The creation of the voxel data structure was partly parallelized, queries were not. All times were measured on an Intel Xenon E5-2665 with 2.4 GHz.

### 3.4.1 Data Structure Creation

Although the creation of the proposed data structure is significantly more expensive than the creation of the *k*-d-tree, the ANN library and the FLANN library, these costs are still within reasonable bounds. Fig. 3.5b compares the creation times for different values of $M_{\max}$. The creation of the Voronoi cells is independent of the value of $M_{\max}$ and thus plotted separately. Fig. 3.5a shows the number of created voxels. They depend linearly on the number of data points, while the choice of $M_{\max}$ introduces an additional constant factor. Note that the constant performance of the proposed method for fewer than $10^5$ data points is based on our particular implementation, which is optimized for large data sets and requires constant time for the creation of several caches.

### 3.4.2 Synthetic Datasets

We evaluate the performance on different datasets with different characteristics. Three synthetic datasets were used and are illustrated in Fig. 3.7. For dataset RANDOM, the points are uniformly distributed in the unit cube $[0,1]^3$. For CLUSTER, points are distributed using a Gaussian distribution. For SURFACE, points are taken from a 2D manifold and slightly disturbed. For each data set, two query sets with 1.000.000 points each were created. For the first set, points were distributed uniformly within the bounding cube surrounding the data point set. The corresponding times are shown in the center column of Fig. 3.8. The second query set has the same distribution as the data set, with the corresponding timings shown in the right column of Fig. 3.8.

The proposed data structure is significantly faster than the simple $k$-d-tree for all datasets with more than $10^5$ points. The ANN library shows similar performance as the proposed method for $M_{\max} = 30$ for the RANDOM and CLUSTER datasets. For the SURFACE dataset, our method clearly outperforms ANN even for smaller point clouds. Note that the SURFACE dataset represents a 2D manifold and thus shows the behavior for ICP and other surface-based applications. Overall, compared to the other methods, the performance of the proposed method is less dependent on the distribution of data and query points. This advantage allows our method to be used in real-time environments.

### 3.4.3 Real-World Datasets

Finally, real-world examples were used for evaluating the performance of the proposed method. Three datasets were collected and evaluated:

*ICP Matching*: Several instances of an industrial object were detected in a scene acquired with a multi-camera stereo setup. The original scene and the matches are shown in Fig. 3.6. We found approximate positions of the target object using the method of Sec. 5 and subsequently used ICP of Sec. 4 for each match for a precise alignment. The nearest neighbor lookups during ICP were logged and later evaluated with the available methods.

*Comparison*: We used the proposed method to find surface defects of the objects detected in the previous dataset. For this, the distances of the scene points to the closest found model were computed. The distances are visualized in Fig. 3.6b and show a systematic error in the modeling of the object.

*ICP Room*: Finally, we used a Kinect sensor to acquire two slightly rotated scans of an office room and aligned both scans using ICP.

The sizes of the corresponding data clouds and the lookup times are shown in Table 3.2. For all three datasets, the proposed method significantly outperforms both our $k$-d-tree implementation and the ANN library by up to one order of magnitude.

Table 3.2: Performance in the real-world scenarios. $|D|$ is the number of data points, $|Q|$ the number of query points. The proposed voxel hash structure is up to one order of magnitude faster than $k$-d-trees, even for large values of $M_{\max}$.

| | | | Voxel Hash, $M_{\max} =$ | | | | |
|---|---|---|---|---|---|---|---|
| **Dataset** | $\|D\|$ | $\|Q\|$ | 30 | 60 | 90 | $k$-d-tree | ANN |
| ICP Matching | 990,998 | 1,685,639 | 0.74 s | 1.04 s | 1.41 s | 12.19 s | 22.0 s |
| Comparison | 990,998 | 2,633,591 | 0.85 s | 1.29 s | 1.87 s | 10.62 s | 232.1 s |
| ICP Room | 260,595 | 916,873 | 0.26 s | 0.37 s | 0.41 s | 0.97 s | 2.5 s |

### 3.4.4 Approximate Method

We conducted several experiments to compare the proposed approach for turning the exact voxel hash method into an approximate method (see Sec. 3.3.6). We varied two parameters of the approximate nearest neighbor structure: The number of voxels in the explicit voxel neighborhood, and the limit on the list length, $L(D, v)$. We allow a neighborhood radius of 1 (using a $3 \times 3 \times 3$ neighborhood of voxels on each level) and 2 ($5 \times 5 \times 5$ neighborhood). We found that larger values have little benefit regarding accuracy but high computational costs. For the list lengths, we evaluated with limits of 1, 5 and 10. We denote the approximate methods with, for example, 2-5 for a voxel neighborhood of 2 and a list length limit of 5.

Table 3.3 compares the different exact and approximate methods regarding data structure creation time, nearest neighbor lookup time and approximation errors. In terms of nearest neighbor lookup times, the proposed approximate method outperforms all other evaluated methods, sometimes by several orders of magnitude. It is the fastest method we know of for comparable error rates, and lookup times scale extremely well with the size of the dataset.

Regarding construction times, the approximate voxel methods are much faster than the exact voxel methods, though still significantly slower than $k$-d-trees, ANN, and FLANN.

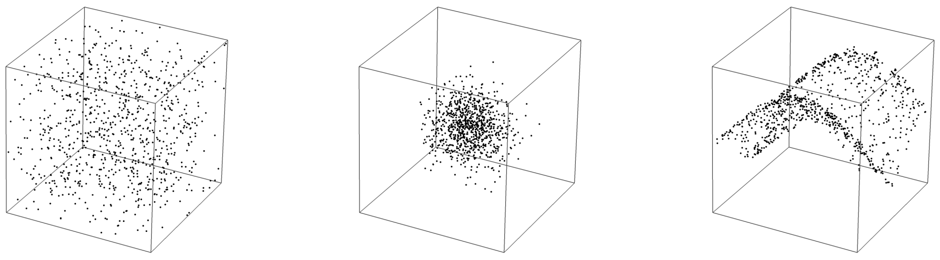
## 3.5 Conclusion

In this chapter we proposed and evaluated a novel data structure for nearest-neighbor lookup in 3D, which can easily be extended to 2D. Compared to traditional tree-based methods, backtracking was made redundant by building an octree on top of the Voronoi diagram. In addition, a hash table was used to allow for a fast bisection search of the leaf voxel of a query point, which is faster than letting the query point descend the tree. The proposed method combines the best of tree-based approaches and fixed voxel grids. We also proposed an even faster approximate extension of the method.

Table 3.3: **Performance of exact and approximate methods on the real-world dataset *ICP Matching*.** Two scenarios are tested: One with a very large data cloud $D$ of approx. one million points (left), one with a much smaller cloud of approx. ten thousand points (right). $t_C$ is the time required for the creation of the data structure, $t_M$ is the time needed to perform around 1.6 million nearest neighbor lookups. Correct is the ratio of points for which the returned nearest neighbor is correct and not just an approximation. Note that the approximate ANN library returned only correct results on this dataset. $e_{\text{mean}}$ and $e_{\text{median}}$ are the errors of the approximate ICP, relative to the diameter of the target object. An error of 1% would indicate that the approximate nearest neighbor has a distance of 1% of the object's diameter to the exact nearest neighbor. Note that for the approximate method 1-1 with a total query time of 0.13 s, the query time per point was only 0.23 $\mu$s.

| | $\lvert D\rvert = 990\,998$ | | | | | $\lvert D\rvert = 13\,333$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Method** | $t_C$ | $t_M$ | Correct | $e_{\text{mean}}$ | $e_{\text{median}}$ | $t_C$ | $t_M$ | Correct | $e_{\text{mean}}$ | $e_{\text{median}}$ |
| $k$-d-tree | 0.16 s | 12.19 s | 100% | 0 | 0 | 1.3 ms | 2.1 s | 100% | 0 | 0 |
| ANN | 0.57 s | 23 s | 100% | 0 | 0 | 17 ms | 26 s | 100% | 0 | 0 |
| FLANN | 0.37 s | 93 s | 12% | 2.4% | 0.73% | 6.1 ms | 2 s | 20% | 2.7% | 0.7% |
| $M_{\text{max}} = 30$ | *hours* | 0.64 s | 100% | 0 | 0 | 18 s | 0.31 s | 100% | 0 | 0 |
| $M_{\text{max}} = 90$ | *hours* | 1.41 s | 100% | 0 | 0 | 6.4 s | 0.43 s | 100% | 0 | 0 |
| Approx. 1-1 | 273 s | 0.37 s | 22% | 1.61% | 0.60% | 0.23 s | 0.13 s | 22% | 1.6% | 0.58% |
| Approx. 1-5 | 305 s | 0.43 s | 35% | 1.35% | 0.35% | 0.31 s | 0.18 s | 41% | 1.2% | 0.23% |
| Approx. 2-1 | 1151 s | 0.47 s | 40% | 0.53% | 0.18% | 0.8 s | 0.15 s | 38% | 0.54% | 0.19% |
| Approx. 2-5 | 1267 s | 0.59 s | 55% | 0.39% | 0.058% | 1.2 s | 0.24 s | 63% | 0.33% | 0 |
| Approx. 2-10 | 1341 s | 0.58 s | 61% | 0.35% | 0.020% | 1.4 s | 0.25 s | 71% | 0.28% | 0 |

The evaluation on synthetic datasets shows that the proposed method is faster than traditional $k$-d-trees, the ANN library and the FLANN library on larger datasets and has a query time that is almost independent of the data and query point distribution. Although the proposed structure takes significantly longer to be created, these times are still within reasonable bounds. The evaluation on real datasets shows that real-world scenarios, such as ICP and surface defect detection, greatly benefit from the performance of the method. The evaluations also showed that the approximate variant of the method can be constructed significantly faster and offers unpreceded nearest neighbor query times.



(a) RANDOM (b) CLUSTER (c) SURFACE

Figure 3.7: Datasets used for the synthetic evaluations. The datasets show different distributions of the target points in $D$: A uniform distribution in a unit cube (RANDOM), a Gaussian distribution forming a cluster of points (CLUSTER), and points sampled from a 2D manifold (SURFACE). We are mostly interested in distributions such as the SURFACE dataset since we typically deal with points on the the surface of 3D objects.

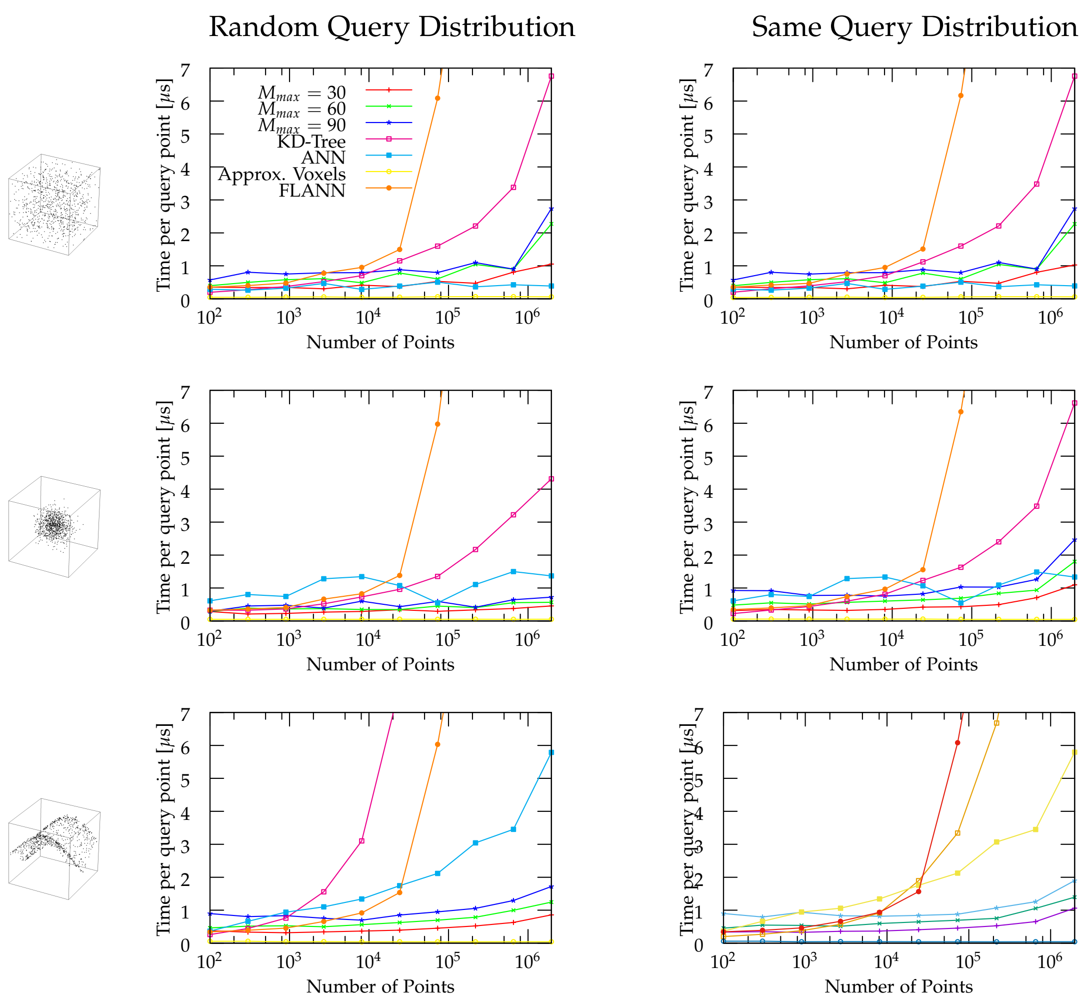


Figure 3.8: Query time per query point for different synthetic datasets and methods. Each row represents a different dataset. From top to bottom: RANDOM, CLUSTER, and SURFACE dataset. The $x$-axis shows the number of data points, *i.e.*, $|D|$, the $y$-axis shows the average query time per query point. For the center column, query points were were randomly selected from the bounding box surrounding the data. For the right column, query points were taken from the same distribution as the data points. Compared to other methods, the query time for the proposed method depends little on the number of data points and is almost independent of the distribution of the data and query points. It is especially advantageous for very large datasets as well as for datasets representing 2D manifolds.

# 4 A Variant of the Iterative Closest Points Algorithm

The object detection step described in Sec. 5 provides only an approximation of the object's pose that is correct only up to several percent of the object's diameter. Many applications, however, such as bin picking or surface defect inspection, require a more accurate pose. This section describes our implementation of the *iterative closest points (ICP)* algorithm, which refines a given initial relative position of two or more point clouds w.r.t. each other. Intuitively, given some initial positions of the point clouds, it minimizes the distances between the surfaces of the point clouds.

The proposed variant of ICP is especially tailored toward the industrial and robotic application scenarios, most notably to be accurate, robust, performant, and to have as few parameters as necessary. For speedup, it employs a coarse-to-fine approach and uses the nearest neighbor lookup structure of Sec. 3. It is especially designed to work with the object detection schemes presented in the subsequent chapters.

This chapter first outlines the basic ICP algorithm. It then describes in detail the design decisions made for the proposed ICP. In Sec. 4.2, the variant that optimizes the position of two clouds relative to each other is outlined. The method is then extended to multiple point clouds in Sec. 4.3.

## 4.1 Introduction and Related Work

ICP was initially introduced by Besl and McKay [16]. As its name already promises, ICP is an iterative method as outlined in Fig. 4.1. Given some initial transformation from the source to the target point cloud, it iterates between correspondence search and distance optimization. The correspondence search finds for each point in the first point cloud the closest point in the second point cloud, given the current transformation. The distance optimization optimizes

```
Input: Scene point cloud $S$
       Model point cloud $M$
       Initial transformation parameters $p_0$
$k \leftarrow 0$
while not converged
  Correspondence Search:
    For each $\mathbf{s} \in S$
      find $\phi(p_k, \mathbf{s}, M) = \arg\min_{\mathbf{m} \in M} |T(p_k, \mathbf{s}) - \mathbf{m}|$
  Minimization:
    $p_{k+1} \leftarrow \arg\min_p \sum_{\mathbf{s} \in S} |T(p_k, \mathbf{s}) - \phi(p_k, \mathbf{s})|^2$
  $k \leftarrow k + 1$
Output: $p_k$
```

Figure 4.1: Outline of the classic ICP (iterative closest points) algorithm with point-to-point metric.

the transformation parameters such that the sum of squared distances between corresponding points is minimized. The method always converges, essentially because in both steps, the sum of squared distances cannot increase [16].

This two-step iteration is a compromise between computational costs and speed of convergence. For convergence, it would be advantageous to optimize correspondences and transformation parameters simultaneously. However, the correspondence search is usually the most expensive part of the method. Using it inside the optimization would require more nearest neighbor lookups, making the method potentially slower. Fitzgibbon [53] used a pre-computed distance transform to do exactly that, showing that it leads to faster convergence and has a wider basin of convergence. However, such distance transforms are – as of now – too expensive for 3D applications.

Today, ICP is a well-studied method with many proposed extensions and modifications affecting speed and robustness. Rusinkiewicz and Levoy [114] give a comprehensive overview for choices regarding point selection, correspondence search and weighting, outlier rejection, error metric, and optimization method. They also show how these choices affect runtime and convergence and propose a real-time variant of ICP.

**Design Choices** In accordance to the objectives mentioned in Sec. 1.2, the design goals of our implementation are accuracy, speed, generality, and few parameters. In addition, it was designed to be extensible to register more than two clouds simultaneously. We also assume that the initial transformation is already within a reasonable distance of the optimal transformation, due to the matching method proposed in Sec. 5. The focus of the method is thus a fast and robust local convergence, rather than a wide basin of convergence. As an overview, we have chosen the following design parameters:

- *Voxel-based nearest neighbors*: We use a voxel-based data structure, as described in Sec. 3, to speed up the nearest neighbor lookups.
- *Tukey-based robust weighting function*: We employ an iteratively reweighted least squares method. For this, the point correspondences are reweighted based on the robust Tukey function, making the method more accurate and robust against outliers.
- *Point-to-plane metric*: We optimize the distance between the points of one point cloud and the planes of another cloud, which are implicitly defined by the normal vectors of the second cloud (Fig. 4.2).
- *Gauss-Newton optimization*: We use an iterative Gauss-Newton method for the optimization, which is parameter free and has fast local convergence. This is of advantage since the initial pose given by the previous detection method is usually already close to the optimum.
- *Numeric derivatives*: The required derivatives w.r.t. rotational parameters are computed numerically, which is faster than analytic derivatives. The derivatives w.r.t. translation are computed analytically.
- *More than two clouds*: The method is designed such that it can register more than two clouds at once.
- *Coarse to fine*: We employ a coarse-to-fine scheme, where fewer points are used int the first iterations of ICP to improve performance.

In the following, we first describe ICP for two point clouds and later extend the method to two more clouds.

## 4.2 Two-Cloud Registration

### 4.2.1 Method

**Model** Intuitively, ICP is supposed to minimize the distance between two or more point clouds $S, M \subset \mathbb{R}^3$, denoted scene and model, respectively. We are interested in obtaining the parameters $p$ of a rigid transformation $T(p) \in \mathrm{SE}(3)$ that transforms scene points $\mathbf{s} \in S$ into model coordinates such that the transformed points $T(p)\mathbf{s}$ are close to the model points. Note that this transformation maps scene to model coordinates instead of vice versa, because this allows to pre-compute a nearest neighbor lookup data structure for the model in an offline stage.

Formally, we model the distance between the two point clouds as the sum of squared distances between the transformed scene points and their closest model point. The target energy we want to minimize is then

$$E(p) = \sum_{\mathbf{s} \in S} w_{\mathbf{s}} d(\mathbf{s}, \phi(p, \mathbf{s}, M))^2 \tag{4.1}$$

where $d$ is the distance metric to be minimized and $\phi(p, \mathbf{s}, M)$ the closest point in $M$ to $T(p)\mathbf{s}$. $w_{\mathbf{s}}$ is a weighting factor that allows a robust refinement in presence of noise and outliers and that is described further below.

**Gauss-Newton Minimization** In order to minimize (4.1) we employ the Gauss-Newton method, an iterative optimization method for least-squares problems [53]. Alternative methods are possible; however, methods such as gradient descent or Levenberg-Marquardt are tailored towards situations where the initial parameters are rather far away from the optimum, while Gauss-Newton has a fast local convergence. This is advantageous for us, since we expect to already have a good approximation from the detection method of Sec. 5. Additionally, alternative methods often require additional parameters for controlling certain step sizes, while the Gauss-Newton method is parameter free.

Following [53], we can re-write the energy (4.1) as

$$E(p) = \sum_{\mathbf{s} \in S} E_{\mathbf{s}}(p)^2, \;\; E_{\mathbf{s}}(p) = \sqrt{w_{\mathbf{s}}} d(\mathbf{s}, \phi(p, \mathbf{s}, M)) \tag{4.2}$$

and define the vector of residuals $e(p) = \{E_{\mathbf{s}}(p)\}_{\mathbf{s} \in S}$ such that $E(p) = |e(p)|^2$. Then,

$$E(p) = e^T e \tag{4.3}$$

$$\nabla E(p) = 2(\nabla e)^T e \tag{4.4}$$

$$\nabla^2 E(p) = 2(\nabla^2 e)e + 2(\nabla e)^T \nabla e \tag{4.5}$$

We denote the Jacobian matrix by $J = \nabla e = (\delta E_s / \delta p_i)_{s,i}$.

We are interested in the update $d$ that minimizes $E(p+d)$. The Gauss-Newton formula, obained by applying the Gauss-Newton approximation $(\nabla^2 e)e \approx 0$ in the Taylor expansion of $E(p+d)$ after setting the derivative of the latter to zero [53], states for the $k$th step,

$$(J^T J) d_k = -J^T e \tag{4.6}$$

$$p_{k+1} = p_k + d_k \tag{4.7}$$

**Parametrization of Update** If the parameter vector $p$ is an overparametrization of the underlying transformation, for example, when using a rotation matrix to parametrize a 3D rotation (see Sec. 2.1), the matrix $J^T J$ might become ill-conditioned. In such cases, it might be of numerical advantage to re-parametrize

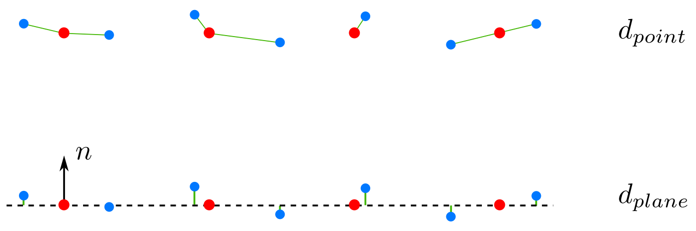


Figure 4.2: Visualization of the point-to-point metric $d_{\text{point}}$ (top) and the point-to-plane metric $d_{\text{plane}}$ (bottom) of two differently sampled clouds (blue, red). Note that even though the alignment is good, $d_{\text{point}}$ adds significant amounts of energy due to the sampling of the red cloud. The distance measure $d_{\text{plane}}$ approximates an infinitely densly sampled red cloud and measures the distances between the blue points and the actual surface from which the red points were sampled.

the update using a vector $\delta$ that has a smaller dimension than $p$. Ideally, the dimension of $\delta$ equals the number of degrees of freedom. Formally, the energy to be optimized is then

$$\hat{E}(\delta_0 + \delta) = E(T(\delta_0 + \delta)p) \tag{4.8}$$

where $T(\delta)p$ applies the updating transformation parametrized by $\delta$ to the parameters $p$. Following (4.6), the update is

$$(J^T J)\delta = -J^T e \tag{4.9}$$
$$p_{k+1} = T(\delta_0 + \delta)p \tag{4.10}$$

We use this re-parametrization mostly for rotations: When parametrizing a 3D rotation in the transformation parameters $p$, it is of advantage to use a parametrization that is easily applicable to vectors, such as quaternions or rotation matrices. However, for these, the update would be overparametrized. Because of this, we use the Rodruiges parametrization for the update $\delta$, which is minimal. Most notably, it is continous around the identity rotation, which is where we expect an updating rotation to be.

The re-parametrization is also used for the refinement of cylinders in Sec. 7.2.4.

**Nearest Neighbor Search** Historically, the computationally most expensive part of ICP is the correspondence search. We use the voxel-based nearest neighbor search introduced in Sec. 3 to speed up the search, using the approximate search introduced in Sec. 3.3.6, which has a slow pre-processing but a fast lookup. This is advantageous for the typical use-case we face, namely searching for an object that is known beforehand such that there is time for expensive pre-processing.

Note that instead of using the voxel hash nearest neighbor lookup, one could also use a $k$-d-tree or other methods. For example, the refinement performed in Fig. 4.3 took 104 ms with an $k$-d-tree and 57 ms with the voxel hash method, a speedup of almost factor 2. The complete detection of the 10 object instances

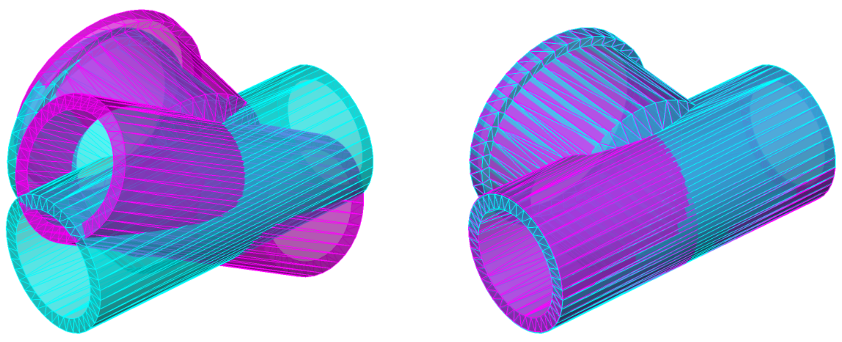


Figure 4.3: Example for ICP: Before refinement (left), after refinement (right). The refinement took 57 ms using the voxel hash method, and 104 ms using a $k$-d-tree for the nearest neighbor lookups. The initialization has a relative distance $m_{1,\text{norm}} \approx 0.28$ from the ground truth.

shown in Fig. 3.6a took 190 ms with the voxel hash method and 320 ms with a $k$-d-tree.

**Metric** Two metrics are common for ICP. The point-to-point metric

$$d_{\text{point}}(\mathbf{a}, \mathbf{b}) = |\mathbf{a} - \mathbf{b}| \tag{4.11}$$

optimizes the distance between points, while the point-to-plane metric

$$d_{\text{plane}}(\mathbf{a}, \mathbf{b}) = (\mathbf{a} - \mathbf{b}) \cdot n(\mathbf{b}) = (\mathbf{a} - \mathbf{b})^T n(\mathbf{b}) \tag{4.12}$$

optimizes the distance between $\mathbf{a}$ and the plane defined by $\mathbf{b}$ and its normal $n(\mathbf{b})$. Our implementation of ICP uses the point-to-plane metric, since it is more robust w.r.t. different sampling distances of the point clouds. In particular, it allows points of one surface to lie in between points of the second surface without additional energy costs, which allows a sparser sampling of the second surface. Fig. 4.2 illustrates this fact. Note, however, that while $d_{\text{point}}$ has a closed-form solution for the update of the parameters [16], $d_{\text{plane}}$ has no such closed-form solution. We thus need to solve the update step using an iterative method.

Note that our implementation is a hybrid, since the correspondences are found based on the point-to-point metric. As a result, the theoretical guarantee of convergence breaks, since the energy might increase when re-computing the correspondences: A point closer in terms of $d_{\text{point}}$ might be further away in terms of $d_{\text{plane}}$. However, we found that this has no noticeable effect in practice, also because we sample the target surface uniformly.

**Parametrization** We parametrize the transformation $T(p)$ as $p = (\mathbf{t}, r) \in \mathbb{R}^3 \times \mathbb{R}^3$. The translation vector $\mathbf{t}$ is used directly, while $r$ is interpreted as Rodruiges-parametrization of the rotation (see Sec. 2.1). The transformation of a scene point $\mathbf{s}$ is then

$$T(p)\mathbf{s} = T((\mathbf{t}, r))\mathbf{s} = R(r)\mathbf{s} + \mathbf{t} \tag{4.13}$$

As described in Sec. 2.1, parametrizations of rotations with 3 parameters always have at least one singular point. In order to avoid this problem, we solve for the *update* $\delta p = (\delta r, \delta \mathbf{t})$ of the transformation instead of the new transformation, such that

$$T(p_k) = T(\delta p)T(p_{k-1}) \tag{4.14}$$

The rotation angle of the update $\delta p$ is typically only a few degrees. This is well within the range where the Rodruiges-parametrization is non-singular.

**Derivatives** In order to compute $J$, we need to calculate the partial derivatives $\frac{\delta E_i}{\delta p_j}$. For the point-to-plane metric and $p = (\mathbf{t}, r)$, we have

$$E_{\mathbf{s}}(p) = \sqrt{w_{\mathbf{s}}}(R(r)\mathbf{s} + \mathbf{t} - \phi(p, \mathbf{s}, M)) \cdot n(\phi(p, \mathbf{s}, M)) \tag{4.15}$$

For convenience, we set $\mathbf{m} = \phi(p, \mathbf{s}, M)$ and obtain

$$\begin{aligned} E_{\mathbf{s}}(p) &= \sqrt{w_{\mathbf{s}}}((R(r)\mathbf{s} - \mathbf{m}) \cdot n(\mathbf{m}) + \ t \cdot n(\mathbf{m})) \\ &= \sqrt{w_{\mathbf{s}}}((R(r)\mathbf{s} - \mathbf{m}) \cdot n(\mathbf{m}) + \ \mathbf{t}_x n(\mathbf{m})_x + \mathbf{t}_y n(m)_y + \mathbf{t}_z n(\mathbf{m})_z) \end{aligned} \tag{4.16}$$

The partial derivatives w.r.t. the components of $\mathbf{t}$ are then simply the weighted components of the normal vector $n(\mathbf{m})$:

$$\frac{\delta E_{\mathbf{s}}}{\delta \mathbf{t}_d} = \sqrt{w_{\mathbf{s}}} n(\mathbf{m})_d, \quad d \in \{x, y, z\} \tag{4.17}$$

In particular, they are constant for a fixed correspondence and thus need to be computed only once per Gauss-Newton optimization step.

The partial derivatives w.r.t. the rotation components in $r$ are more complicated to derive analytically, since they include the derivative of the matrix $R(a, \alpha)$ (compare (2.2)). We found that instead of using the analytical derivative, using a first-oder numerical approximation is more practical. Formally, we compute

$$\begin{aligned} \frac{\delta E_{\mathbf{s}}}{\delta r_x} &= (E_{\mathbf{s}}(p + (0,0,0,\delta_{\text{rot}},0,0)) - E_{\mathbf{s}}(p))/\delta_{\text{rot}} \\ \frac{\delta E_{\mathbf{s}}}{\delta r_y} &= (E_{\mathbf{s}}(p + (0,0,0,0,\delta_{\text{rot}},0)) - E_{\mathbf{s}}(p))/\delta_{\text{rot}} \\ \frac{\delta E_{\mathbf{s}}}{\delta r_z} &= (E_{\mathbf{s}}(p + (0,0,0,0,0,\delta_{\text{rot}})) - E_{\mathbf{s}}(p))/\delta_{\text{rot}} \end{aligned} \tag{4.18}$$

where in practice, we use $\delta_{\text{rot}} \approx 0.1$.

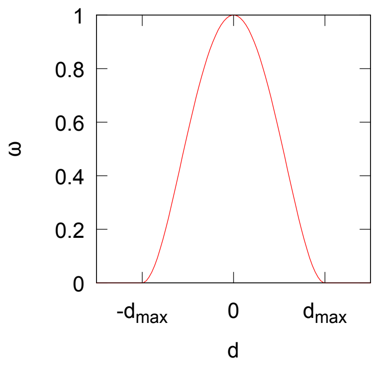


Figure 4.4: Weighting function for the robust refinement (see (4.19)). Close correspondences are weighted higher, correspondences further away than $d_{\max}$ have a zero weight.

**Robust Refinement** An important aspect of refinement algorithms is the robustness against noise and outliers. The quadratic factor in the least-squares formulation leads to a strong influence of outliers onto the final result. A common approach to reduce this influence is to use the weighting factors of (4.1) to reduce the impact of outliers: The larger the distance between two corresponding points, the lower their influence on the result. The weights are re-computed after some iterations, making it a *Iteratively reweighted least squares* (IRLS) method [122, 66].

We compute the weights based on some maximum distance $d_{\max}$ between two corresponding points. Given a correspondence $c$ with distance $d$ between the two corresponding points, the weight is

$$w_c = w(d) = \begin{cases} (1 - d^2/d_{\max}^2)^2 & \text{if } d < d_{\max} \\ 0 & \text{else} \end{cases} \tag{4.19}$$

The weighting function $w(d)$ is based on the Tukey Biweight [10, 134, 97], and is visualized in Fig. 4.4.

The parameter $d_{\max}$ is initialized using some a-priori information about the expected accuracy of the initial transformation. For example, for the rigid 3D matching of Sec. 5, this accuracy can be estimated based on some of the used sampling rates.

If the convergence stalls, $d_{\max}$ is reduced in order to remove additional outliers. For this, we first fit a Gaussian distribution into the current set of distances $D$. This is done efficiently by computing the median of $D$ and using the *median absolute deviation* to estimate the mean as

$$\sigma \approx 1.4826\, median(D) \tag{4.20}$$

Note that this assumes that the median of the signed distances is approximately zero and that the noise on the distances is of Gaussian nature. We set the new $d_{\max}$ to $2\sigma$, which – again assuming Gaussian noise – covers around 95% of the correspondences. This way, strong outliers are suppressed, as their distance is likely larger than $2\sigma$, and thus their weight will be set to zero.

```
Input: Scene point cloud $S$
       Model point cloud $M$
       Initial transformation parameters $p_0$
       Maximum distance $d_{\max}$

for $k$ from 1 to max_num_steps
  Correspondence Search:
    For each $\mathbf{s} \in S$
      Find $\phi(p_k, \mathbf{s}, M) = \arg\min_{\mathbf{m} \in M} |T(p_k, \mathbf{s}) - \mathbf{m}|$
      Compute weight $w_{\mathbf{s}}^k = \text{tukey}(|T(p_k, \mathbf{s}) - \phi(p_k, \mathbf{s}, M)|)$

  Perform Gauss–Newton optimization to compute $p_k$

  if GN–update not successful
    Compute new $d_{\max}$ using (4.20)
    If $d_{\max}$ did not change much
      return

Output: $p_k$
```

Figure 4.5: Outline of the full ICP algorithm. The correspondences are kept fixed, and several iterations of Gauss-Newton are performed to optimize the parameters w.r.t. the current correspondences. If the update step failed to improve the result, the distance threshold is adjusted, which effectively removes noisy and outlier points from the optimization.

### 4.2.2 Experiments

We performed several synthetic experiments to measure the behavior of the proposed ICP w.r.t. noise and clutter, as well as its basin of convergence. For this, several thousand scenes were rendered from random viewpoints, and ICP was performed with different settings of noise, initialization, clutter, and number of scene points. We measured the relative error using the $m_{1,\text{norm}}$ metric, as explained in Sec. 2.2.4, (2.37).[1] Additionally, we show the error in rotation and the relative error of the translational components of the final pose.

Three objects were used: The Stanford bunny [124], the pipe joint (see Fig. 3.6) and a simple box. The box shows a typical failure case of ICP: In scenes where only two sides of the box are visible, ICP is unable to recover one degree of freedom of the transformation. Since we measure over several thousand scenes, many of them will have only two sides visible, leading to a significantly lower accuracy of the box compared to the other two objects. The bunny is rather

[1] As reminder, $m_{1,\text{norm}}$ is the maximum distance of a surface point from its ground truth location, relative to the diameter of the object.

```
Input: Correspondences $C$
       Scene point cloud $S$
       Previous transformation $T(p)$

For $j$ from 1 to max_num_gn_steps
  Compute residual vector $(E_{\mathbf{s}})_{s \in S}$ using (4.2)
  Compute Jacobian using (4.17) and (4.18)
  Solve update $(J_j^T J_j) d^j = -J_j^T e_j$ (see (4.6))
    using LU–decomposition of $J^T J$
  $p_k^j = p_{k-1} + d^j$
  Compute energy $E_j^k = E^T E$

Find best step $j^* = \arg\min_j E_j^k$
Return $p_k^{j^*}$
```

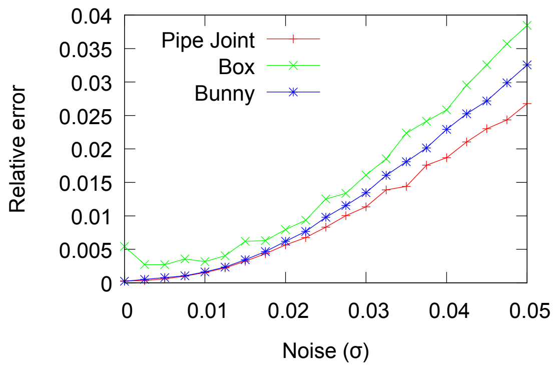
Figure 4.6: Outline of the Gauss-Newton update step for ICP. The parameters are updated iteratively for several steps, and the corresponding energy is computed after each step. Note that a Gauss-Newton step is not guaranteed to improve the result. Because of this, we compute the actual energy after each step, and return the best parameters.

well-behaved, showing a distinctive geometry and no symmetries. The joint is somewhat in between: Its large cylindrical part has several symmetries, which are broken by the smaller joint. Depending on the viewpoint, the symmetry-breaking joint might be visible or not.

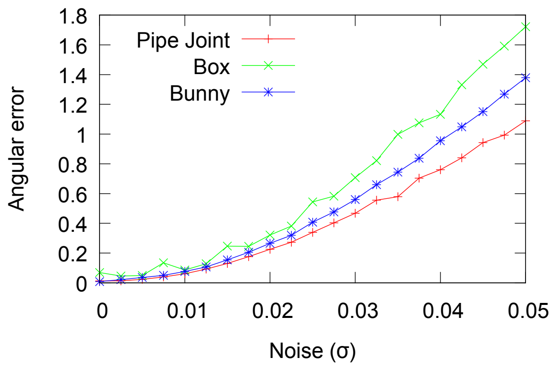
Fig. 4.7 shows the result of adding **Gaussian noise** to the $z$-component of the individual points of the rendered scenes. This models typical noise of well-calibrated range sensors, where noise in the measurement of $z$ exists, while $x$ and $y$ are recovered correctly from the calibration. The amount of noise is measured relative to the diameter of the model: $\sigma = 0.03$ means that the Gaussian noise had a standard deviation of 3% of the model's diameter.

Fig. 4.8 shows the results of adding random, additional **clutter** points to the scenes. The clutter points were put into the bounding box of the target object, meaning that they were rather close to the target shape.

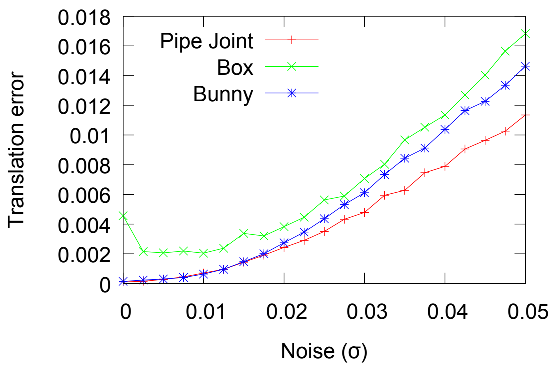
In Fig. 4.9, we measure the **basin of convergence**: The method is initialized with transformations that are increasingly different from the ground truth. We measure the deviation of the initial transformation from the ground truth using the relative error measurement $m_{1,\text{norm}}$. The bunny and the joint show clear and large basins of convergence, where relative errors up to $\approx 0.2$ are successfully compensated. The box shows the afore-mentioned behavior. To get a visual intuition of the relative distances, Fig. 4.3 shows an example of an initialization with $m_{1,\text{norm}} \approx 0.28$.



(a) Noise vs. relative error



(b) Noise vs. rotational error

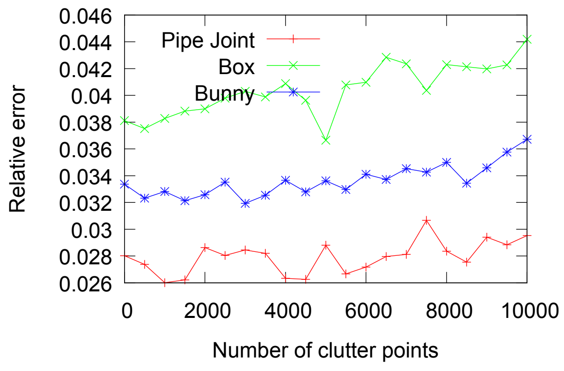


(c) Noise vs. translational error

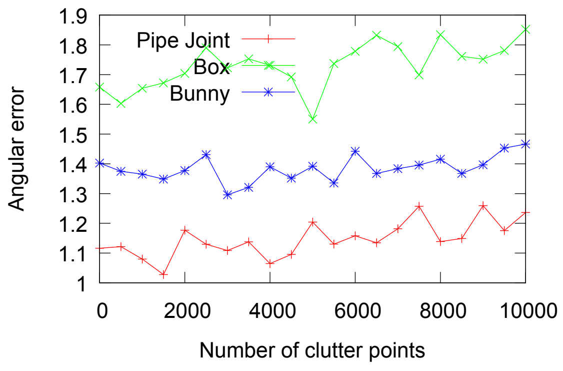
Figure 4.7: Influence of Gaussian noise on the accuracy of ICP. The amount of Gaussian noise added to each point is measured relative to the diameter of the model. The relative error $m_{1,\text{norm}}$ is shown, the error in the rotational component (in degrees) and the relative error in the translational component of the result w.r.t. ground truth. The graphs show the results of several thousand artificially rendered scenes. Note how geometrically diverse objects, such as the pipe joint and the bunny, behave much better than an object like the box, which has many self-similar, planar sides.

Note that the exact errors w.r.t. noise and clutter also depend on the number of points in the scene that form the object. More points allow the method to recover a more accurate pose, as statistical effects with start to cancel the errors out. Fig. 4.10 shows how the number of points in the scene affects the overall results: More points lead to a more robust result, since there will be more points to compensate the errors.

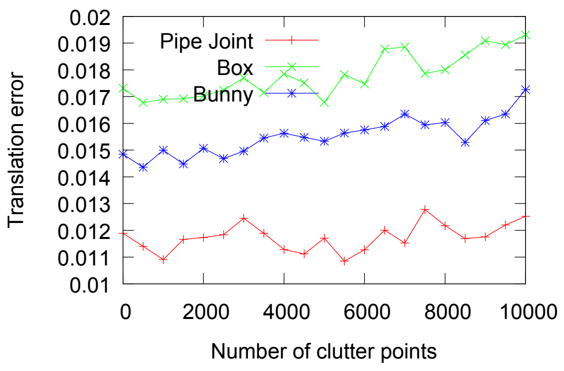
To conclude, the accuracy of the proposed ICP depends on the scene point noise, the number of scene points, and the shape of the object, while the influence of clutter points is effectively reduced.



(a) Clutter vs. relative error



(b) Clutter vs. rotational error



(c) Clutter vs. translational error

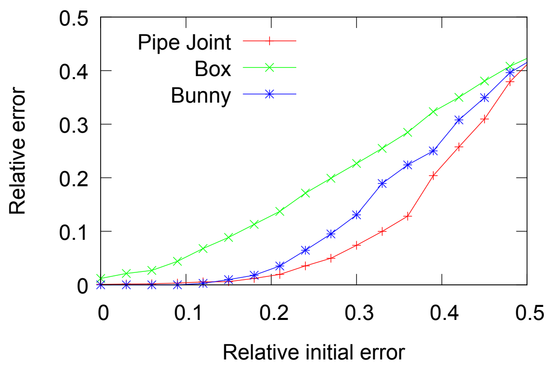
Figure 4.8: Influence of clutter points on the accuracy of ICP. Gaussian noise with $\sigma_{\text{rel}} = 0.05$ was added to each scene. Several thousand artificial scenes were rendered and a certain amount of additional clutter points were added to each. The points were added within the bounding box of the target shape, *i.e.*, rather close to the object. The robust reweighting of the proposed ICP successfully compensates even for large amounts of clutter, such that the results show only a small trend upwards.

## 4.3 Multi-Cloud Registration

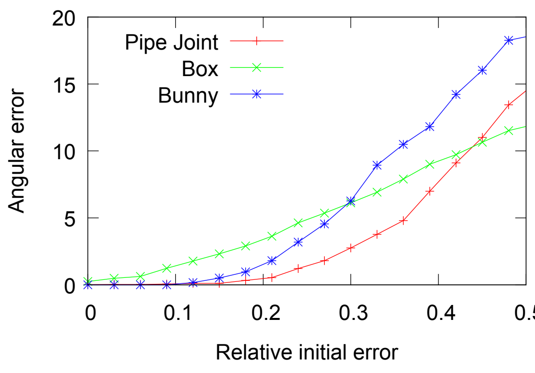
Some applications require a simultaneous pose optimization of more than two point clouds. For example, an object can be reconstructed from multiple, partially overlapping scans from different directions. If the precise position of the scanner is unknown, one can first approximate the relative poses between the scans through pairwise registration and then optimize all positions simultaneously in a global manner.

### 4.3.1 Method

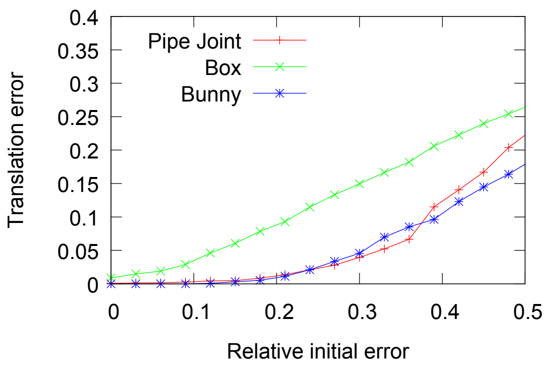
It is straightforward to extend the ICP method described above to the registration of more than two point clouds. To avoid overparametrization, we fix the pose of one of the clouds and compute the other poses relative to that fixed cloud. Additionally, the energy function must be adapted, since now the position of



(a)



(b)



(c)

Figure 4.9: Influence of accuracy of the initial transformation on the result of ICP. The figures show the accuracy of the results w.r.t. the accuracy of the initial transformation. The graphs show aggregated results over several thousand scenes. Overall, relative errors up to $\approx 0.2$ were successfully compensated. Note that the box, which consists of many planar sides, is more difficult: If only one or two sides of the box are visible, it is impossible for ICP to recover one degree of freedom.

both points of a correspondence can change.

**Notation** We are given a set of $M+1$ point clouds $S_0, S_1, \ldots S_M$ and a set of $M$ initial transformations $T_1^0, T_2^0, \ldots T_M^0 \in \mathrm{SE}(3)$ of all but the first cloud. The transformations map the clouds into the coordinate space of the first cloud. Similar to the two-cloud ICP, we parametrize the transformations in a single vector $p$ that concatenates translation and rotation components of all clouds. We write $p_i = (\mathbf{t}_i, r_i)$ for the parameters of cloud $i$. $T_i$ is the transformation corresponding to $p_i$, with

$$T_i\mathbf{x} = R(r_i)\mathbf{x} + \mathbf{t}_i \tag{4.21}$$

For simplicity of notation and to avoid over-complicated if-else-cases, we will sometimes write $p_0$ for cloud 0, even though that transformation is fixed to be identity. The number of parameters is $6M$, $3M$ for the rotations and $3M$ for the

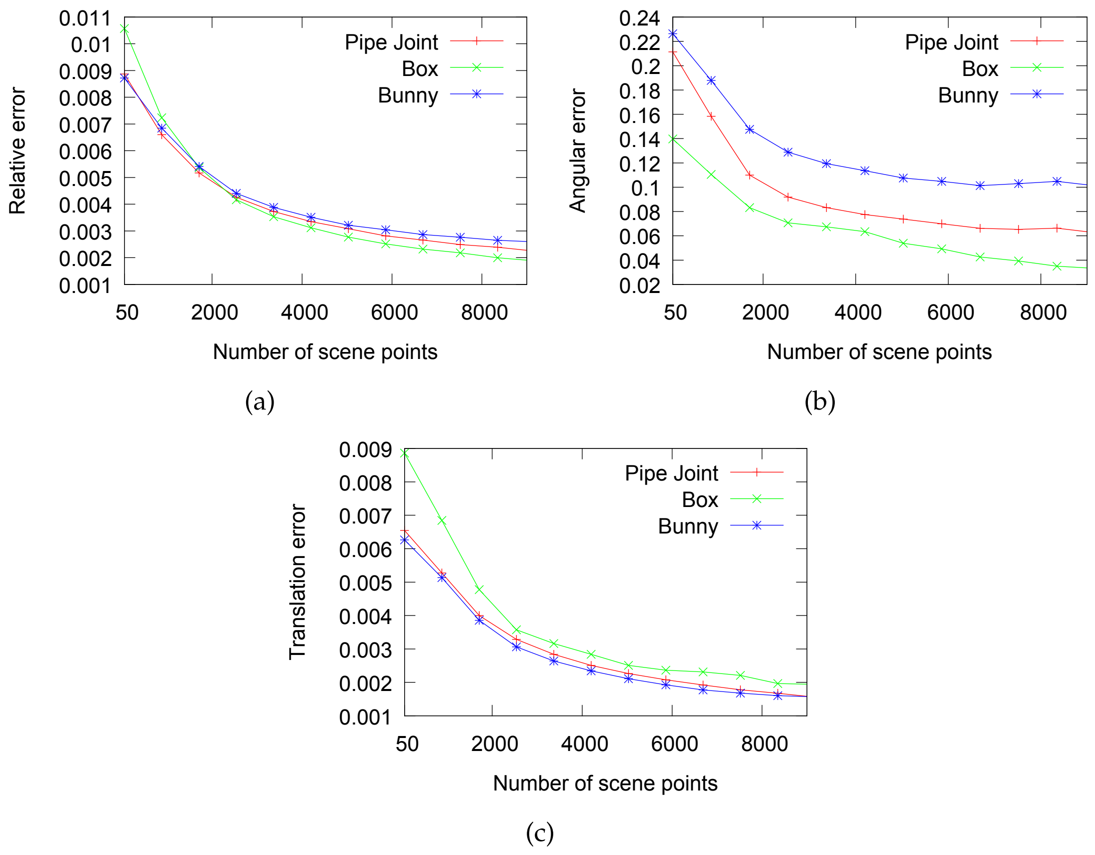


Figure 4.10: Number of scene points vs. final accuracy of ICP. Gaussian noise with $\sigma_{\text{rel}} = 0.01$ was added to each scene. As expected, the more points are available, the more accurate the refined pose will be.

translations.

**Nearest Neighbor Search** A common use-case of the above two-cloud ICP is to refine the position of a fixed object in varying scenes, for example, a machine vision system used to detect a single object only. In this case, it makes sense to use the voxel hash data structure from Sec. 3 for nearest neighbor search, which has a more expensive pre-processing phase but processes nearest neighbor queries much faster. In the case of multi-cloud ICP, though, a common use-cases is to refine the positions of multiple scans of a scene only once. In this case, a tradeoff must be found for the time invested in the data structure creation and time used for the nearest neighbor queries.

Such a tradeoff is difficult to find analytically. Fig. 4.11 shows examplary timings for data structure creation and alignment. Overall, for setups where the data structure creation is not separated from the refinement phase, *k*-d-trees perform faster. If an offline phase is available, voxel hashes are faster.

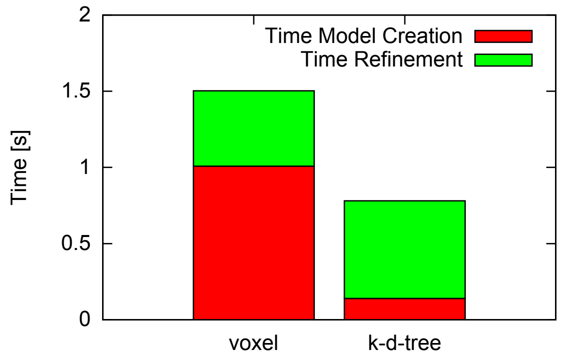


Figure 4.11: Timing for non-parallelized multi-cloud ICP on three point clouds. Using the voxel data structure leads to a faster on-line phase (495 ms for voxels, 641 ms for *k*-d-trees) but a slower data structure creation (1 second for voxels, 140 ms for *k*-d-trees).

**Correspondence search** For each point $\mathbf{a} \in S_i$ in one of the clouds and all other clouds $S_j, j \neq i$, the corresponding closest point $\mathbf{b} = \phi(p, \mathbf{a}, S_j)$ is searched. We denote by $c = (\mathbf{a}, \mathbf{b}) \in C$ the set of all found correspondences. As for the two-cloud ICP, we ignore corresponding points if their distance is larger than the current threshold $d_{\max}$.

Note that in the worst case, such as all clouds being equal, the number of correspondences is $|C| \in \mathcal{O}(SM^2)$, where $S$ is the average point cloud size, $S = \frac{1}{(M+1)} \sum_{j=0}^{M} |S_j|$.

**Energy** Given a correspondence $c = (\mathbf{a}, \mathbf{b})$ between a point $\mathbf{a} \in S_i$ and its closest point $\mathbf{b} = \phi(p, \mathbf{a}, S_j) \in S_j$, the energy function for that correspondence reads

$$\begin{aligned} E_c(p) &= \sqrt{w_c}\, d_{\text{plane}}(T_i\mathbf{a}, T_j\mathbf{b}) \\ &= \sqrt{w_c}\, \big((T_i\mathbf{a} - T_j\mathbf{b}) \cdot (R(r_j)n(\mathbf{b}))\big) \\ &= \sqrt{w_c}\, \big(((R(r_i)\mathbf{a} + \mathbf{t}_i) - (R(r_j)\mathbf{b} + \mathbf{t}_j)) \cdot (R(r_j)n(\mathbf{b}))\big) \end{aligned} \tag{4.22}$$

The global energy function is then

$$E(p) = \sum_{c \in C} E_c(p)^2 \tag{4.23}$$

**Derivatives** The derivatives of $E_c$ w.r.t the translational parameters can be computed analytically from (4.22):

$$\left.\begin{aligned} \tfrac{\delta E_c}{\delta \mathbf{t}_{i,d}} &= \sqrt{w_c}(R(r_j)n(\mathbf{b}))_d \\ \tfrac{\delta E_c}{\delta \mathbf{t}_{j,d}} &= -\sqrt{w_c}(R(r_j)n(\mathbf{b}))_d \end{aligned}\right\} \quad d \in \{x, y, z\}$$

The rotational derivatives w.r.t. $r_i$ and $r_j$ are computed numerically, similar to the case of only two clouds (see (4.18)). The derivatives w.r.t. translation and rotation parameters of all the other clouds are 0.

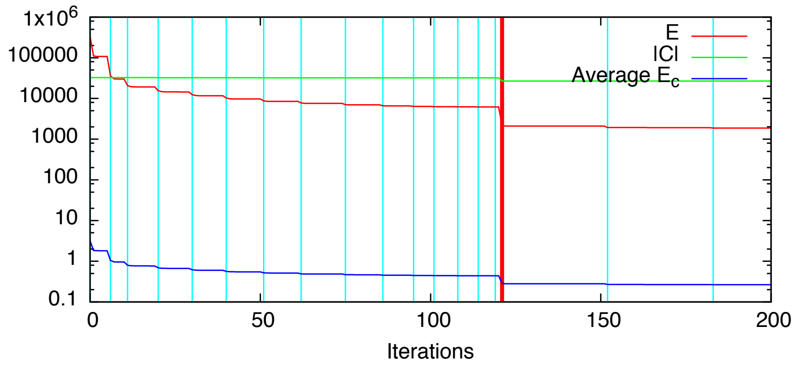


Figure 4.12: Example of the development of the total energy $E$, the number of correspondences $|C|$ and the average energy (distance) per correspondence, $\sqrt{E/|C|}$ during a multi-cloud ICP with four point clouds. The vertical lines in cyan indicate where the correspondences were re-computed. In the steps in-between, iterations of the Gauss-Newton optimization are performed. The vertical line in red indicates where the distance threshold $d_{\max}$ was adapted. Overall, the average distance dropped from 3.2 to 0.25.

**Avoiding the Explicit Computation of the Jacobian** The Jacobian $J$ of the multi-cloud optimization has the dimensions $|C| \times 6M$. As discussed above, the worst case for $|C|$ is $|C| \in \mathcal{O}(SM^2)$, such that the size of $J$ is $\mathcal{O}(SM^3)$. However, $J$ is sparse, since at most 12 entries – 6 for both clouds – for each correspondence are non-zero.

To avoid excessive memory usage, we compute $J^TJ$ and $J^Te$ inline by iterating over the correspondences and updating the corresponding entries of $J^TJ$ and $J^Te$. The processing time for this is $\mathcal{O}(|C|)$, which is bound by $\mathcal{O}(M^2)$. Note that the worst case occurs only if all clouds share significant overlap. $J^TJ$ is of size $6M \times 6M$ and $J^Te$ of size $6M$, which is in practice orders of magnitudes smaller than $J$.

**Robust Refinement** In order to reduce the influence of outliers, the same weighting scheme as for the two-cloud refinement is used, with the only difference that for the update of $d_{\max}$, the correspondences between *all* point clouds are considered, instead of only the correspondences between two clouds.

### 4.3.2 Experiments

The base method of the multi-cloud ICP is identical to that of the single-cloud ICP. In order to rule out any additional negative effects, Fig. 4.13 shows the resulting accuracy for simultaneously registering 13 synthetic point clouds. Overall and as expected, there is a direct correlation between the input and output noise.

Fig. 4.14 shows a real-world example of the multi-cloud method: 15 stereo scans of a scene were taken and aligned pairwise. The resulting chain of transfor-

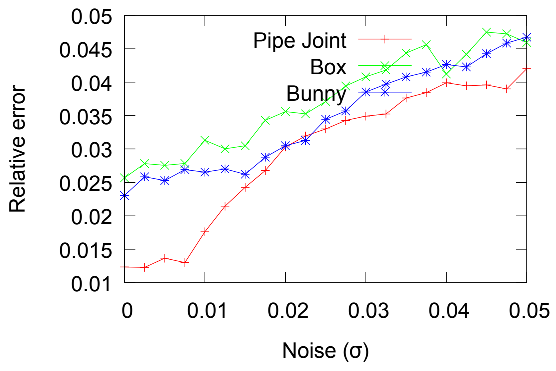


Figure 4.13: Influence of noise on the accuracy of the multi-cloud ICP. Three different objects, pipe joint, box and bunny, were rendered from 13 directions. The initial transformations were slightly disturbed, and the multi-cloud ICP was run on all clouds simultaneously. Gaussian noise was added to the point positions of the point clouds, with varying $\sigma_{\text{rel}}$, which is given relative to the diameter of the corresponding model.

mations was used to initialize the multi-cloud ICP, which produced the presented result. The multi-cloud ICP took around 4 seconds.

## 4.4 Parallelization

The runtime of ICP is dominated by the nearest neighbor search and the computation of the $J^T J$ and $J^T e$. A parallelization of both steps of the ICP algorithm is straightforward. It allows to use all available CPU cores with very little synchronization overhead.

- For the correspondence search, each thread processes a subset of the points. No synchronization between the threads is necessary.

- For the in-line computation of $J^T J$ and $J^T e$, each thread processes a subset of correspondences and updates a local copy of the matrices. The matrices are combined after all threads are finished. No synchronization between the threads is necessary. Since $J^T J$ and $J^T e$ are only of size $6M \times 6M$ and $6M$, respectively, the overhead for storing and combining the results of different threads is negligible.

Note that for the rigid object detection pipeline in Sec. 5, however, ICP was not parallelized. Instead, multiple candidates are tracked by different threads, effectively moving the parallelization to a higher level.

## 4.5 Coarse-to-Fine

The ICP method is usually applied to the complete set of scene points. Depending on the sensor resolution, some ten- to hundred thousand points might be included. We found, however, that a much smaller number of scene points is usually sufficient to already significantly increase the accuracy of the object's position. The upside of using fewer points is that ICP is significantly faster, since both of the most time-consuming steps – finding nearest neighbors and computing the $J^T J$ and $J^T e$ – are linear in the number of points. Fig. 4.15a illustrates this fact.

We thus propose to use a coarse-to-fine approach for ICP, where the pose is first refined on a coarse level, using only few scene points. The result is used to initialize the refinement on the next finer level, using more scene points, which hopefully requires fewer iterations owning to the more accurate initialization. This step is repeated until the full number of scene points is reached.

We choose the number of scene points to be used in each step such that they form an approximate geometric progression between a minimum number of scene points, $n_{\min}$, and the actual number of scene points $|S|$:

$$N = (n_{\min}, \alpha n_{\min}, \alpha^2 n_{\min}, \ldots, |S|) \tag{4.24}$$

For example, for $n_{\min} = 130$ and $|S| = 10000$ and four scene levels,

$$\alpha = \sqrt[3]{\frac{10000}{130}} \approx 4.25 \tag{4.25}$$

ICP would thus be performed on sampled scenes with the approximate number of points

$$N = (130, 550, 2350, 10000) \tag{4.26}$$

Several strategies are possible for selecting the number of levels, such as using a fixed number of levels or using a fixed progression factor $\alpha$.

Fig. 4.15b shows the result when using a four-step approach on the same dataset used for Fig. 4.15a. Only a single iteration of ICP is performed per level, initialized with the result of the previous level. While the relative error between the regular and the coarse-to-fine approach is comparable, there is a significant speedup when using the coarse-to-fine approach. For larger scenes, the coarse-to-fine approach can be over 3 times faster compared to ICP on a single level.

**Rigid Object Detection Pipeline** Looking ahead a bit, Sec. 5 will introduce a voting scheme for detecting rigid objects in 3D point clouds that returns approximate poses of the object. Those approximations are then refined using the ICP proposed in this chapter, using a two-step coarse-to-fine approach: The first refinement is performed on a sparsely sampled version of the original scene, while the second refinement is performed using all scene points.

This approach has three major advantages:

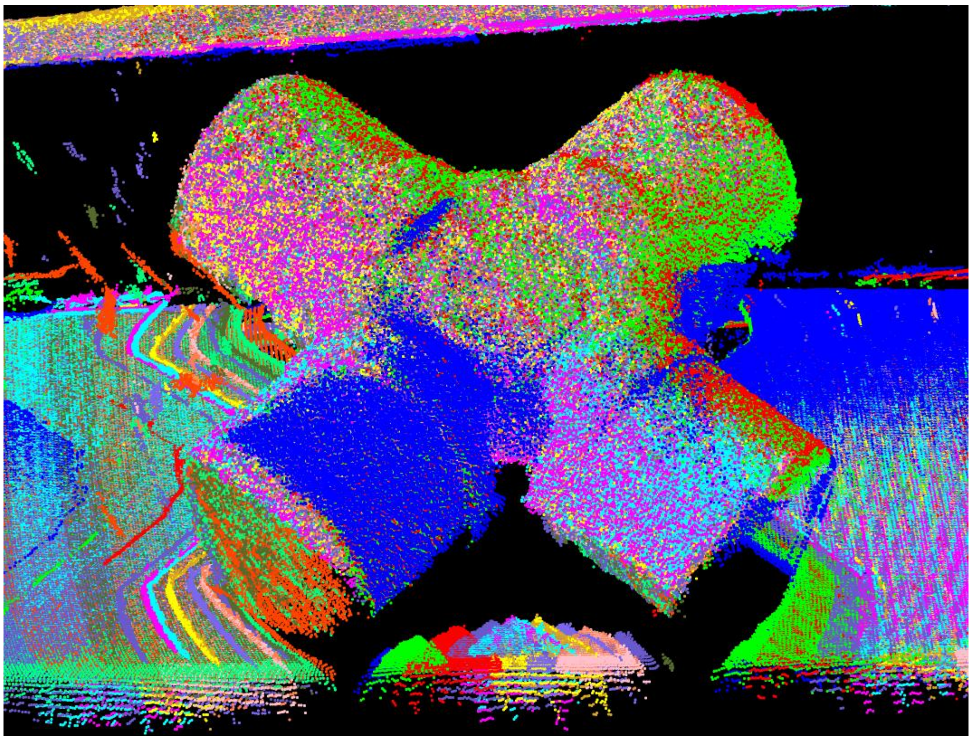
1. *More accurate quality rating*: Sparsely refined poses allow a more accurate estimation of the quality or score of the pose. The number of votes of a pose is subject to noise, for example because of sampling and scene noise. Even a fast run of ICP on a sparsely sampled scene allows a more accurate computation of the quality of a pose. This in turn allows the method to concentrate on better poses for subsequent processing first, leaving other poses for later.

2. *More accurate pose comparison*: Sparsely refined poses allow for an improved non-maximum-suppression in the pose space. The voting can lead to several similar poses, all of which represent the same underlying correct pose. Sparse refinement makes them move closer to the ground-truth pose, which in turn allows to remove all of them once the correct pose was found.

3. *Speedup of subsequent refinement*: Similar to the coarse-to-fine approach above, since the initial pose is more accurate, less iterations of the slow refinement that uses more or all scene points are required.

## 4.6 Conclusion

This chapter introduced a variant of the Iterative Closest Points algorithm for refining an approximate pose of a rigid object, designed to be accurate, robust, performant, and to have as few parameters as necessary. It was specifically designed to refine the pose of an approximate object detector, trading a smaller basin of convergence for faster and more robust refinement. The refinement speed is optimized by using the voxel-based nearest neighbor method and by employing a coarse-to-fine approach on multiple sampling levels of the scene point cloud. The method was finally extended to allow refinement of more than one point cloud simultaneously. The experiments show that the method is fast and highly robust against noise and clutter points.


(a) Aligned point clouds. Different colors represent different clouds.

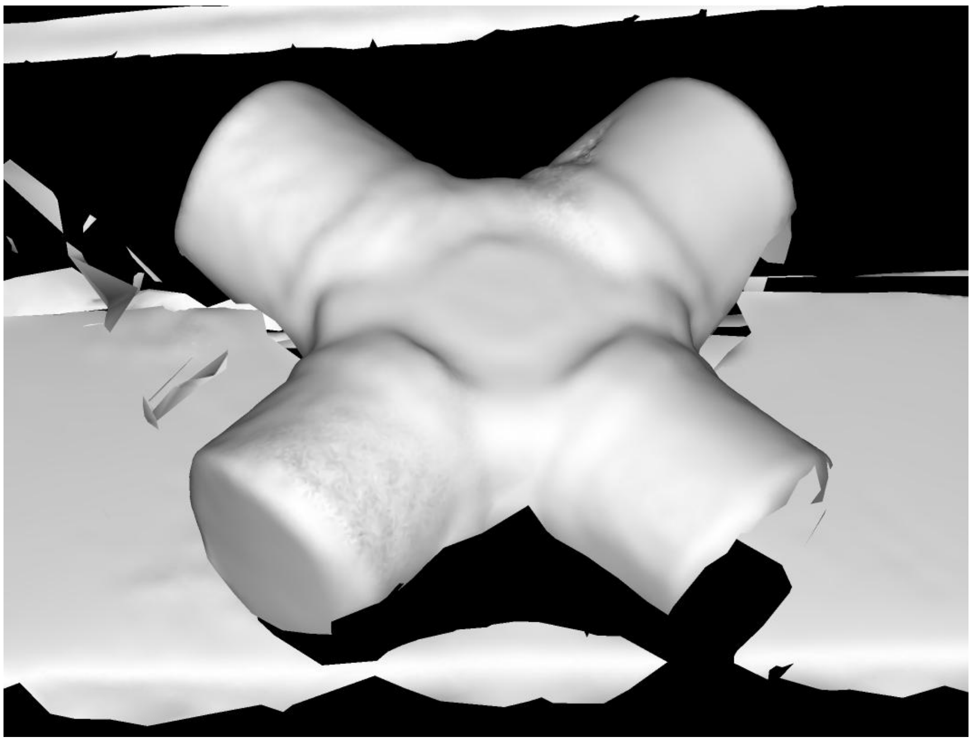
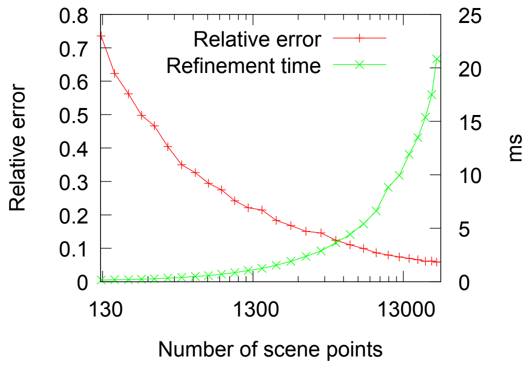
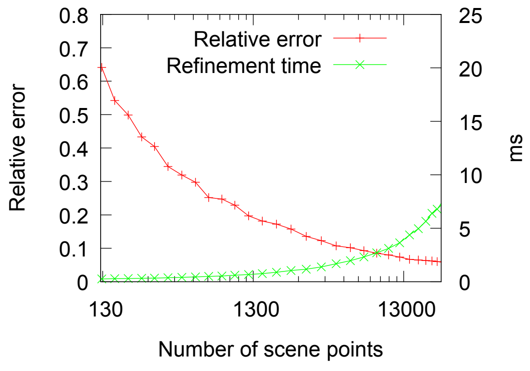

(b) Triangulated result

Figure 4.14: Exemplary result of multi-cloud ICP. 15 scans of a scene were taken using a stereo system. The scans were aligned pairwise sequentially to obtain initial transformations, and the resulting transformations were globally aligned using the presented multi-cloud ICP.



(a) ICP on full scene only



(b) Coarse-to-fine approach

Figure 4.15: Motivation of coarse-to-fine refinement. (a): Influence of the number of scene points on the refinement accuracy and refinement runtime. The values were measured on an artificial scene with Gaussian noise. As one would intuitively expect, using more scene points leads to a higher accuracy at the price of higher runtimes. (b) A coarse-to-fine approach of ICP, where first a fast refinement is performed with only few scene points. The result is then used to initialize ICP on more scene points, but with fewer of the more expensive iterations. Here, a total of four such differently sampling densities was used. While the accuracy is comparable to the full ICP in (a), runtime is significantly improved, especially for large scenes.

Part III

# Matching in 3D Point Clouds

This part introduces four novel object detection methods for different kind of modalities, object classes and transformation types. All four methods are build on the same framework: A Hough-Transform like voting scheme that uses a data-driven, local parametrization. For this, the object's location is parametrized relative to a fixed point of the observed scene that is assumed to be on the surface of the object. The voting scheme uses point pair features that describe the geometric relation of two points.

The four detection methods differ in the details of the parametrization, of the feature and of the voting. The following table gives an overview over the methods and their differences. While the details are given in the corresponding chapters, this table serves as an overviewing guide. It is recommended to start reading with Sec. 5, which explains the baseline method in detail. The other three chapters are then independent of each other.

| Method | Rigid *(§5)* | Multimodal *(§6)* | Primitives *(§7)* | Deformable *(§8)* |
|---|---|---|---|---|
| Input Data | 3D point cloud | RGB-D data | 3D point cloud | 3D point cloud |
| Transformation | Rigid | Rigid | Rigid | Deformable |
| Parameters | local *(§5.3.3)* | local | reduced local + shape *(§7.2.2)* | local |
| Features | 3D point pairs *(§5.3.1)* | Multimodal point pairs *(§6.3.1)* | 3D point pairs | 3D point pairs |
| Model | hash table *(§5.3.2)* | hash table | Implicit or hash table *(§7.2.3)* | hash table |
| Voting | 1 round *(§5.3.4)* | 1 round | 1 round | $\geq 1$ rounds, using graph *(§8.2.4)* |

The basline method has several advantages over prior art, such as speed, robustness, accuracy and a certain optimality, all of which are inherited by the other three methods.

# 5 Rigid Object Detection in 3D Point Clouds: A Local Voting Scheme

One of the challenges in 3D computer and machine vision is the detection and localization of rigid 3D objects in 3D point clouds. Such a localization is an important step in many applications. For example, the position of the objects can be fed to a robot for manipulation, or the 3D data can be used to inspect the object for completeness. This chapter introduces a novel approach to 3D object detection. At its core, it is a **voting scheme** that uses **point pair features** on a **restricted, data-driven, local parameter space**. This voting scheme is integrated into a detection pipeline that removes duplicate poses and further refines the results.

The evaluations on multiple synthetic and real-world datasets show that it is a fast, robust and generic object detector. Combined with a refinement scheme, such as the ICP variant of Sec. 4, it is also highly accurate.

Parts of this chapter previously appeared in [46].

## 5.1 Introduction

The recognition of free-form objects in 3D data obtained by different sensors, such as laser scans, time-of-flight cameras and stereo systems, has been widely studied in computer vision [22, 87, 91]. Global approaches [76, 108, 110, 123, 150, 156] are typically neither very precise nor fast, and are limited mainly to the classification and recognition of objects of certain type. By contrast, local approaches that are based on local invariant features [15, 31, 40, 55, 75, 92, 111, 115, 126, 133, 139, 35] became extremely popular and proved to be quite efficient. However, defining local invariant features heavily depends on local surface information, which is directly related to the quality and resolution of the acquired scene and model data.

In contrast to the approaches outlined above, we propose a method that is a hybrid between local and global approaches. The model is described in a global manner, by describing all pairs of surface points using point pair descriptors, and by collecting all those pairs in a global model description. Matching, or pose recovery, is done locally, by using a voting scheme on a locally restricted parameter space: After fixing one scene point, only those poses are considered for which that scene point lies on the object surface. The point pair feature describes the relative position and orientation of two oriented points. The global model description consists of all model point pair features and represents a mapping from the feature space to the model, where similar features on the model are grouped together. Such a representation provides a global distribution of all point pair features on the model surface. Compared to the local methods, which require dense local information, our approach allows the model and the scene data to be represented only by a sparse set of oriented points that can easily be computed from the data of most 3D input devices and methods. Using sparse data also allows for an important increase in the recognition speed, without significant decrease in the recognition rate. A fast voting scheme, similar to the Generalized Hough transform [7], is used to optimize the model pose in a locally reduced search space that is parametrized in terms of points on the model and rotation around the surface normals.

We test our approach on a number of synthetic and real sequences and compare it to state-of-the-art approaches. We demonstrate that in the presence of noise, clutter, and occlusions we obtain better results than Spin Images [75] and Tensors [92] in terms of recognition rates and efficiency.

## 5.2 Related Work

The problem of detecting and recognizing free-form objects in three-dimensional point clouds is well studied. Methods such as ICP [158] optimize a coarse registration locally (see also Sec. 4). In contrast, we are only interested in methods for object registration that do not need an approximate pose as input.

**Global Methods** Several global methods for 3D object detection have been proposed. However, they either detect only parametrized shapes such as planes, cylinders and spheres, or require an a priori segmentation of the scene. Several approaches detect objects using a variant of the Generalized Hough Transform [76, 110, 156] but are limited to primitive objects because the recovery of a full 3D pose with 6 degrees of freedom is computationally too expensive. Schnabel *et al.* [123] detect primitives in point clouds by using an efficient variant of RANSAC. Park *et al.* [108] detect objects in range images by searching for patterns of the object created from multiple directions. They parallelize their algorithm on the GPU in order to obtain matching times of around 1 second. By contrast, our approach works with general 3D point clouds and is more efficient on the CPU without parallelization.

**Local Methods** A second class of methods, local methods, usually use a pipeline that first identifies possible point-to-point correspondences between the model and the scene. Multiple correspondences are then grouped to recover the pose of the model.

One way of finding the correspondences is the use of *local point descriptors* or *local surface descriptors* that describe the surface around a certain point using some low-dimensional representation. The descriptors must be as discriminating as possible while being invariant against a rigid movement of the surface, robust against clutter and noise, and embedded in a framework that can deal with occlusion. Point correspondences are built by comparing the descriptors of the model to those of the scene, usually using some high-dimensional nearest neighbor retrieval. Most of the local methods concentrate on the design of the local surface descriptor. This might be motivated by the encouraging results of local 2D image descriptors such as SIFT [85], which have been applied very successfully over the last decades. Extensive surveys over different descriptors are given in [22, 87, 91] and more recently in [62], which is why only a few selected ones are covered here.

Point descriptors can be categorized by the radius of influence that affects them. Local descriptors exploit the geometric properties around a surface point, most notably by using different representations of the surface curvature [15, 40] or by fitting polynomials [111] or splines [35]. Regional descriptors try to capture the surface shape around the reference point. Splashes [133] describe the distribution of normal orientations around a point. Point Signatures [31] use the distance of neighboring points to the normal plane of the reference point. Point Fingerprints [139] use geodesic circles around the reference point as description. Gelfand *et al*. [55] introduced an integral descriptor for point cloud registration that describes the intersecting volume of a sphere around the reference point with the object. Rusu *et al*. [116] proposed the *Persistent Point Feature Histogram* over all neighboring points of the reference point. They later improved the computational performance [115] and included viewpoint-dependent statistics [117]. Their histograms can then be used for finding point correspondences or for surface classification. Chen and Bhanu [24] introduced a local surface patch representation that is a histogram of shape index values vs. the angle between normals. Johnson and Hebert [75] introduced spin images, which are histograms of the surface created by rotating a half-plane around the normal of the reference point and summing the intersecting surface. In [126], spin images are used as an index into a database of objects with subsequent pose detection using a batch RANSAC. Yamany and Farag [154] described surface signatures, which can be seen as variant of the spin images where the cylindrical coordinates are replaced by a combination of point-to-point distance and the angle between the points and the normal vector. Ruiz-Correa *et al.* [113] defined a symbolic surface signature to detect classes of similar objects. Taati and Greenspan [140] proposed a descriptor with variable dimension, using an optimization approach to select the most discriminative geometric features. Tombari *et al.* [144, 119] introduced the SHOT descriptor, which combines histogram-based and signature-based descriptors. The descriptor was also extended to include texture information [145]. Steder *et al.* [130] proposed the *normal aligned radial feature* (NARF), a feature point detector and descriptor that is based on discontinuities of range images. They explicitly remove shadow corners and veil points, which are smoothing artifacts between two depths. Their method, however, does not immediately generalize to generic 3D point clouds.

Somewhat different from the previous approaches, Bariya and Nishino [8] proposed a scale-invariant local 3D shape descriptors. Note however that since 3D measurements are usually calibrated, there is usually no need to match over multiple scales compared to 2D images. A corner detector is applied over multiple scales, and the scale with the maximum response is used to define a local scale. They report runtimes in the range of minutes.

A distinctive disadvantage of all of the afore mentioned local methods is that they require dense 3D data that is not too noisy in order to extract a robust and discriminative local descriptor. We found that in many practical applications, the 3D information does not meet these criteria. The data is often too sparse because of low-resolution sensors[1] or non-dense reconstruction methods (such as stereo on surfaces with little texture). Other acquisition methods produce data that is too noisy or suffers from quantization effects. Additionally, in order for the local feature descriptors to have a chance to be discriminative, the surface parts within the descriptor's radius of influence must be different on different parts of the object. This is often not the case for objects that have large amounts of self-similarity or symmetry. Increasing the radius of influence increases the discrimination capabilities of feature descriptors but makes the descriptors more sensitive to missing surface parts and clutter.

The method proposed in this chapter does not have any of those requirements. Instead, the data can be sparse and noisy. Also, while self-similarities of the object's surface make the proposed method slower, it does not affect the detection rate. Also, we do not require a post-processing step such as RANSAC for grouping the correspondences found by feature points.

**Point-Pair Based Approaches** Our method uses *point pair features* that describe the relative position and orientation of two oriented points, *i.e.*, two points with normal vectors. Oriented points are sometimes called *surflets* [150], as they can be

[1]Commercial time-of-flight sensors have resolutions of only 200x200 pixels.

seen to describe a small, planar patch of the surface. The following is an overview of other methods where such 3D point pairs were used.

A first group of methods uses *histograms of point pairs* as local or global descriptor. Hetzel *et al.* [64] use local, viewpoint dependent histograms of normal and curvature distributions to describe objects. They identify segmented objects through histogram comparison between scene and database, allowing accurate and fast identification of objects. Similar, but in a more global manner, Wahl *et al.* [150] use global histograms of point pairs to identify segmented 3D shapes. They also discretize the point pairs and compare the resulting histograms to robustly identify objects in a few milliseconds. Compared to the propose method, both approaches do not identify the object's pose and require the object to be segmented. Rusu *et al.* [116] use local histograms of point pairs to create robust, invariant local surface descriptors. They also present a way for robustly extracting a robust radius of influence, as well as a way for quickly approximating the local point pair histogram. Their method is designed for the registration of large-scale scenes, such as multiple scans of a room. Stein and Medioni [133] use point pairs to build a repeatable local coordinate frame, and to build local surface descriptors called *splashes* or *super splashes*. Additionally, they used pairs of splashes similar to point pairs, using their distance and angles to measure spatial consistency between scene and model splashes. Their approach is local and therefore suffers from the same drawbacks that were mentioned above.

A second group of methods uses the fact that a matching point pair is in general enough to establish a rigid transformation between two 3D point clouds. The methods differ in how the point pairs are matched and how potential matches are grouped. Stockman [136] uses matching sets of two planar patches and the connecting edge, called SSE configuration, to generate pose hypotheses. A subsequent pose clustering method collects the hypotheses. While this approach matches the same features we do, it lacks the efficient local voting scheme and the constant-time feature lookup. Matei *et al.* [88] measure point pairs similar to us, by directly using angles and distances. However, they lack a fast indexing and accessing scheme, as well as a corresponding voting scheme. Instead, random point pair features from both the scene and the model are tested for similarity, and matches are used to build transformation hypotheses. Winkelbach *et al.* [152] match two point clouds by using a *birthday attack* on point pair features: Random point pairs are selected iteratively from both clouds, and stored in a hash table. On average, only $1.25n$ point pairs need to be selected until a matching pair is found. Their approach is a particularly interesting application of the birthday attack [6]. However, contrary to our approach, it is non-deterministic and might be computationally expensive if many similar point pairs exist or if the surface overlap is small. Contrary to this, our method has a deterministic runtime and always returns – up to certain statistical effects of noise – the globally optimal pose. Mian *et al.* [92] use two reference points to define a coordinate system where a three-dimensional tensor is built by sampling the space and storing the amount of surface intersecting each sample. The tensors are stored using a hash table that allows an efficient lookup during the matching phase. The tensors can be used not only for matching, but also for general point cloud registration. Compared to our approach, their method is significantly slower and has a worse detection rate. Skotheim *et al.* [128, 146] proposed a method where a single point pair in the model is selected and searched for in the scene. Several criteria ensure that the pair is discriminative (*i.e.*, as unique as possible) and stable. However, the selection of the point pair requires a manual step. The approach is also sensitive to noise and occlusion of one of the two points of the point pair.

## 5.3 Rigid Object Detection in 3D

While the Hough transform was successfully applied to 2D detection problems, its application to 3D was so far limited. This is mostly due to the different properties of the feature and the parameter space in 3D compared to 2D, which are both much larger and have a more complex geometry. The differences of the two spaces are in particular:

1. The parameter space for rigid 3D transformations is much larger than in 2D. Instead of 3 degrees of freedom – two for translation, one for rotation – it contains 6 degrees of freedom. This makes a naïve Hough transform orders of magnitude more expensive, as the accumulator space grows exponentially with the number of parameter dimensions. Note that even for 2D problems, the accumulator space is sometimes deemed to be too large and methods for its reduction were proposed [148].

2. The 3D parameter space is more complex, mostly because of the geometry of SO(3), the manifold of 3D rotations (see Sec. 2.1). It is, in particular, more expensive to compute the inverse kernel for some feature, *i.e.*, the list of all parameters that explain a feature, since it requires operations with 3D rotations. It is also difficult to find a discretization of SE(3) into approximately equally sized bins.

3. In 2D, features are ordered nicely in a grid array of pixels. This allows efficient, constant-time lookup for a feature at a particular position. Contrary to this, 3D point clouds are typically sparse, as they describe a 2D manifold in 3D. Even though indexing data structures exist that speed up the lookup of a feature at a particular position, they still require certain compromises and are less efficient than in 2D.

However, 3D offers a distinctive advantage over 2D: In 2D images, the outline or contour of an object – which defines its position – is not measured directly. Instead, it must be extracted using some feature extraction step, such as a line or edge detector. Contrary to this, 3D sensors measure points on the physical surface of the scene. Thus, no additional feature extraction is required to extract the object's boundaries, aiding the method's robustness.

**Overview** This section introduces a local voting scheme that detects rigid objects in 3D point clouds. It uses three building blocks, which are described in the following sections.

1. *Point pairs* are the features the method is based upon. They are described using a low-dimensional descriptor, and stored in a hash table for fast retrieval.
2. *Local parameters* describe the position of a rigid object relative to some given scene point. They are used to reduce the size of the parameter space.
3. A *voting scheme* that uses point pairs to recover the optimal local coordinates, *i.e.*, the rigid transformation that aligns the most model and scene points.

The voting scheme is embedded into a detection pipeline which is outlined in Fig. 5.17 and Sec. 5.3.9. It might be useful to reference this figure while going through the following sections, which discuss the different steps in more detail.

**Notation** In the following sections, we write $M \subset \mathbb{R}^3$ for the point cloud describing the model that is to be found, and $S \subset \mathbb{R}^3$ for the observed scene in which the model is to be found. Points from those clouds are usually denoted as $\mathbf{m} \in M$ and $\mathbf{s} \in S$, respectively. Both clouds are oriented, *i.e.*, there is a normal vector $n(\mathbf{s})$, $n(\mathbf{m})$ assigned to each point.

### 5.3.1 3D Point Pair Features

**Definition** For two points $\mathbf{m}_1$ and $\mathbf{m}_2$ with normals $\mathbf{n}_1$ and $\mathbf{n}_2$, we set $\mathbf{d} = \mathbf{m}_1 - \mathbf{m}_2$ and define the point-pair feature $\mathbf{F}$ as

$$\mathbf{F}(\mathbf{m}_1, \mathbf{m}_2, n(\mathbf{m}_1), n(\mathbf{m}_2)) = (\|\mathbf{d}\|_2, \measuredangle(\mathbf{n}_1, \mathbf{d}), \measuredangle(\mathbf{n}_2, \mathbf{d}), \measuredangle(\mathbf{n}_1, \mathbf{n}_2)) \tag{5.1}$$

where $\measuredangle(\mathbf{a}, \mathbf{b}) \in [0, \pi]$ denotes the angle between two vectors (see Sec. 2). $\mathbf{F}$ is asymmetric w.r.t. exchanging the two points. Our feature avoids defining a local coordinate frame, such as in [152]. Instead, angles and distance are measured directly, thus minimizing the effect of noise in the normal directions and point locations. To avoid notational clutter, we will often simply write $\mathbf{F}(\mathbf{m}_1, \mathbf{m}_2) = \mathbf{F}(\mathbf{m}_1, \mathbf{m}_2, n(\mathbf{m}_1), n(\mathbf{m}_2))$, or even simply $\mathbf{F}$ if the two points are clear from context. Fig. 5.1 illustrates the feature.

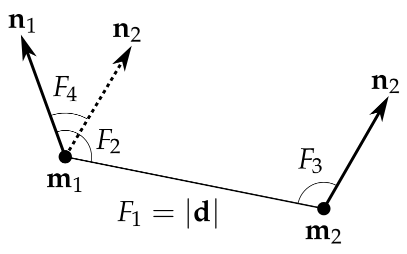


Figure 5.1: Point pair feature $\mathbf{F}$ of two oriented points $\mathbf{m}_1$ and $\mathbf{m}_2$ with normals $\mathbf{n}_1$ and $\mathbf{n}_2$. The component $F_1$ is the distance of the points, $F_2$ and $F_3$ are the angles between the normal vectors and the difference vector $\mathbf{d}$, and $F_4$ is the angle between the two normals.

**Mirroring** Note that $\mathbf{F}$ is invariant against mirroring the complete system, *i.e.*,

$$\mathbf{F}(\mathbf{m}_1, \mathbf{m}_2, \mathbf{n}_1, \mathbf{n}_2) = F(-\mathbf{m}_1, -\mathbf{m}_2, -\mathbf{n}_1, -\mathbf{n}_2) \tag{5.2}$$

Since mirroring changes the handedness of the system, mirrored pairs of points can in general not be transformed onto each other with a rigid transformation. The feature can be extended such that it includes the handedness of the system defined by $\mathbf{n}_1$, $\mathbf{n}_2$ and $\mathbf{d}$. For example, the two normal vectors can be used to construct a unique direction $\mathbf{v}$, and that direction can be compared with the difference vector:

$$\mathbf{v} = \mathbf{n}_1 \times \mathbf{n}_2 \tag{5.3}$$

$$s = \operatorname{sgn}(\mathbf{v} \cdot \mathbf{d}) \tag{5.4}$$

Equivalently, one can use the sign of the determinant of the system matrix

$$s = \operatorname{sgn} \begin{bmatrix} \mathbf{n}_1 & \mathbf{n}_2 & \mathbf{d} \end{bmatrix} \tag{5.5}$$

to define a feature $\mathbf{F}_m$ that is not mirror-invariant,

$$\mathbf{F}_m(\mathbf{m}_1, \mathbf{m}_2) = (\|\mathbf{d}\|_2, \measuredangle(\mathbf{n}_1, \mathbf{d}), \measuredangle(\mathbf{n}_2, \mathbf{d}), \measuredangle(\mathbf{n}_1, \mathbf{n}_2), s) \tag{5.6}$$

However, we found that in practice the additional computation costs for computing and using $\mathbf{F}_m$ are not worth the additional discriminativeness of the feature. The number of point pairs that are mirrored w.r.t. each other is low enough to not affect detection performance too much.

**Properties and Discussion** Point pair features have several unique and properties that make them especially suited for the matching applications we have in mind. First, $F$ is *fast to compute*, as no extensive pre-processing (such as smoothing or regular triangulation) of the point cloud is necessary. Note that the potentially expensive step of computing normal vectors is required only for the subsampled scene, not for the full scene point cloud. The features are *invariant against rigid*

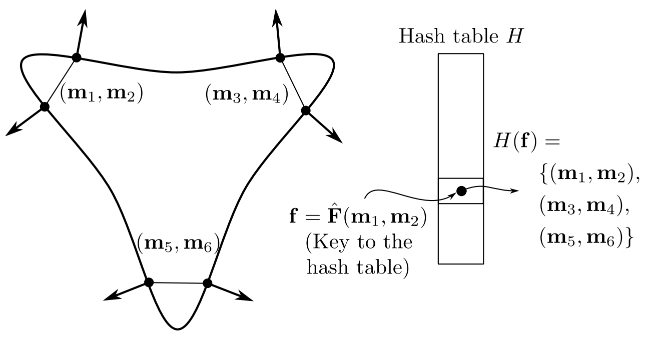

$(\mathbf{m}_1, \mathbf{m}_2)$
$(\mathbf{m}_3, \mathbf{m}_4)$
$(\mathbf{m}_5, \mathbf{m}_6)$
Hash table $H$
$\mathbf{f} = \hat{\mathbf{F}}(\mathbf{m}_1, \mathbf{m}_2)$
(Key to the hash table)
$H(\mathbf{f}) = \{(\mathbf{m}_1, \mathbf{m}_2), (\mathbf{m}_3, \mathbf{m}_4), (\mathbf{m}_5, \mathbf{m}_6)\}$

Figure 5.2: Illustration of the point pair database using a hash table. Three similar point pairs on the model (left) are grouped into the same bin of the hash table (right).

*transformations*, making methods based on them invariant as well. We will show in Sec. 5.3.2 that because of its small dimension, it is also *fast to match*. Except for the case of mirrored configurations and for rare degenerated configurations[2], two matching point pairs can be used to construct a *unique rigid transformation* that maps both points and their normal onto each other. Point pairs are also *far-reaching*, non-local, thus allowing matching of points far (in the sense of the target object's size) away from each other.

Finally, with some exceptions, point pairs are quite *discriminative*, and a specific point pair configuration often repeats only few times on an object. Exceptions to this are objects that are (partially) symmetric. For example, point pairs repeat rather often on large planar patches, on spheres and on cylindrical surfaces. The last paragraph of the following section elaborates further on this matter. In Sec. 7, we will modify the detection scheme presented in this chapter to deal with symmetric primitive shapes.

### 5.3.2 Point Pair Matching

A crucial step of the proposed object detection methods is the matching of scene point pairs to model point pairs: Given two points in the scene, we need to quickly identify all pairs of points on the model that have a similar distance and orientation. Depending on factors such as the dimension of the feature vector, the size of the feature database and the expected noise levels, feature matching can be a rather time-consuming step. Since the $\mathbf{F}$ has a small dimension, and since the expected level of noise on the components of the feature is fixed, we can use a direct hashing approach. Hashing is among the fastest methods for feature lookup and allows us to find all matching model point pairs in constant time, independent of the size of the feature database.

The distance and the angular components of $\mathbf{F}$ are sampled in steps of $d_{\text{dist}}$

[2]For example, if both normals and $\mathbf{d}$ are linearly dependent.

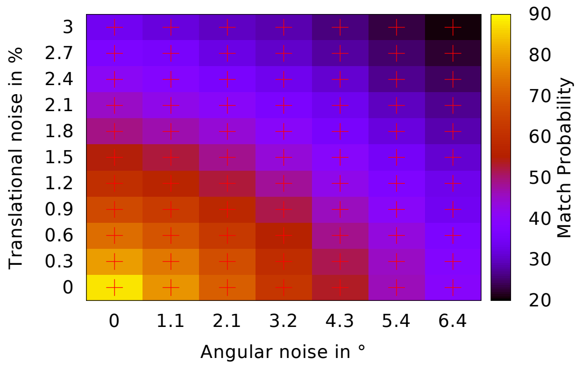


Figure 5.3: Illustration of the binning problem. Gaussian noise with the shown standard deviations (in % of the diameter of the model for translation, in degrees for the normal) was added to both the point positions and the normal angles. The more noise the points and the normals have, the less likely it is that a point pair is correctly matched against the database. Note that the probability of a matching point pair is less than 100% even for no noise, owing to the different sampling of scene and model. The sampling of the feature was $d_{\text{dist}} = 3\%$ and $d_{\text{angle}} = 12°$.

and $d_{\text{angle}} = 2\pi/n_{\text{angle}}$, respectively

$$\hat{\mathbf{F}} = \left( \left\lfloor \frac{F_1}{d_{\text{dist}}} \right\rfloor, \left\lfloor \frac{F_2}{d_{\text{angle}}} \right\rfloor, \left\lfloor \frac{F_3}{d_{\text{angle}}} \right\rfloor, \left\lfloor \frac{F_4}{d_{\text{angle}}} \right\rfloor \right) \in \mathbb{Z}^4 \tag{5.7}$$

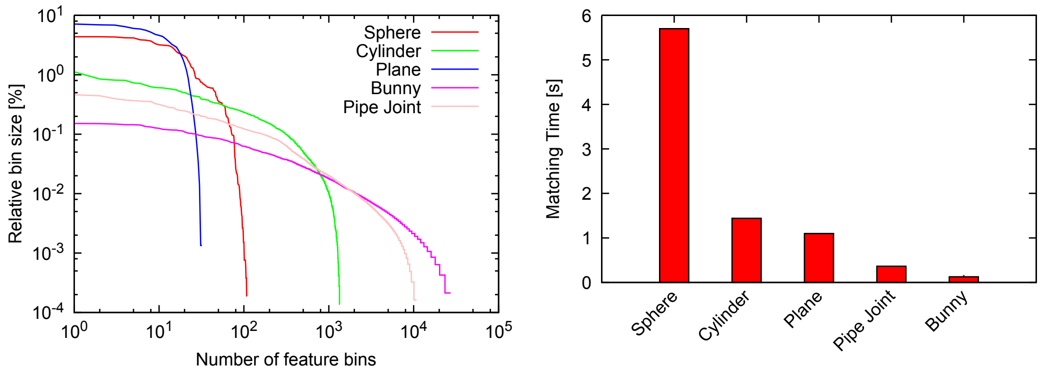
The four non-negative integer values of $\hat{\mathbf{F}}$ are used as index into a hash table $H$ that stores the list of all feature vectors whose discrete versions are identical. Formally, it is a mapping

$$H : \mathbb{Z}^4 \to M^2, \;\; H(\mathbf{f}) = \{(\mathbf{m}_1, \mathbf{m}_2) \in M^2 : \hat{\mathbf{F}}(\mathbf{m}_1, \mathbf{m}_2) = \mathbf{f}\} \tag{5.8}$$

Fig. 5.2 illustrates this lookup. The hash table $H$ thus allows to find all model point pairs similar to a given scene point pair in constant time.

**Query Range** Direct hashing is only reasonable for fixed-range nearest neighbor queries. For point pairs, we can define that range based on the sampling of the model and the expected noise of point positions and normal angles in the scene. In practice, we set $n_{\text{angle}} = 30$ such that $d_{\text{angle}} = 12°$. By default, $d_{\text{dist}}$ is set to 3% of the model's diameter, however, that value can be adjusted based on sensor noise.

**Binning Problem** Note that directly hashing the feature is among the fastest methods for feature lookup, promising constant lookup times. The downside of this method, however, is the binning problem: One or more components of the feature vector might be close to the discretization boundary, and noise



(a) (b)

Figure 5.4: (a) Distribution of the point pair features for different shapes. Shapes with high amounts of self-similarity or symmetry, such as spheres, planes, or cylinders, tend to have many identical features. Shapes with a more distinctive geometry, such as the bunny, have more distributed point pair features. The distribution of the pipe joint, which is built of several cylinders, is somewhere in between. (b) Timings when detecting an object in a scene that contains only itself. The less uniform the distribution of the feature list sizes are, the longer the matching takes.

can move the component – and thus the discretized feature – into a different, incorrect bin. The binning problem is especially problematic with higher feature dimensions, since the probability of being close to and potentially crossing a hash cell boundary increases with each dimension.

We found that thanks to the small dimension of $\mathbf{F}$, binning is not much of a problem in the case of point pair features. Fig. 5.3 shows the probability that a point pair is not matched correctly, based on the amount of translational and rotational noise and using the discretization parameters mentioned above. Note that, as shown later, our method is able to detect the object even if only $10\% - 20\%$ of the features are matched correctly. Note also that strategies exist to mitigate the binning problem, either during creation of the data structure, by adding model point pairs to multiple neighboring bins of the database, or during the lookup phase, by evaluating neighboring bins.

**Distribution** The length of each list $H(\mathbf{f})$ depends on how many similar features exist on the model's surface. Fig. 5.4a illustrates the distribution of the lengths of the lists $H(\mathbf{f})$ for different object types. For asymmetric free-form objects, such as the clamp and the bunny, list lengths show a long-tail distribution with no outliers. For objects with high self-similarity, such as planes, cylinders, and spheres, the distribution is biased towards a few large and many small lists. Sec. 5.4.3 evaluates how the feature distribution affects the matching performance.

### 5.3.3 Local Coordinates

The unbounded and complex geometry of SE(3) makes it difficult to use a Hough transform for recovering a rigid 3D transformation. This section describes how a sub-manifold of SE(3), the *local coordinates*, can be used to reduce the dimension and complexity of the parameter space. Thanks to this reduction, we will be able to employ a Hough transform for detecting generic, free-form objects in 3D.

**Idea** The fundamental idea of local coordinates is to take one observed, oriented 3D point, the *reference point* $\mathbf{s}_r \in S$ and its normal $n(\mathbf{s}_r)$, and to restrict oneself to those rigid transformations that explain $\mathbf{s}_r$. In other words, we allow only those rigid transformations for which $\mathbf{s}_r$ is on the surface of the transformed model. The set of those parameters can be seen as the inverse kernel of $\mathbf{s}_r$:

$$K_M^{-1}(\mathbf{s}_r) = \{T \in \mathrm{SE}(3) : \mathbf{s}_r \in T\,M\} \tag{5.9}$$

More intuitively, we allow the object to move into all positions for which $\mathbf{s}_r$ is a part of the object's surface, and for which $n(\mathbf{s}_r)$ is perpendicular to the object's surface.

**Parametrization** There is a natural parametrization for the set of local coordinates w.r.t. $\mathbf{s}_r$. By definition, for each local coordinate, there exists some point $\mathbf{m}_r \in M$ on the object's surface that corresponds to $\mathbf{s}_r$. In other words, each $T_M \in K^{-1}(\mathbf{s}_r)$ aligns $\mathbf{s}_r$ and its normal with a model point $\mathbf{m}_r$ and its normal:

$$T\mathbf{m}_r = \mathbf{s}_r \tag{5.10}$$

$$R(T)n(\mathbf{m}_r) = n(\mathbf{s}_r) \tag{5.11}$$

We call $\mathbf{m}_r$ the *corresponding point* of $\mathbf{s}_r$ on the object's surface. Aligning $\mathbf{s}_r$, $\mathbf{m}_r$, and their normals leaves one degree of freedom: the object can still be rotated around the normal vector of $\mathbf{s}_r$. We use another angle $\alpha$ to describe this rotation.

Summed up, every 3D transformation $T \in K^{-1}(\mathbf{s}_r)$ can be parametrized by a model point $\mathbf{m}_r \in M$ and a rotation angle $\alpha \in [0; 2\pi[$, or

$$T = T(\mathbf{m}_r, \alpha) \in M \times [0; 2\pi[ \tag{5.12}$$

**From Local Parameters to Global Parameters** At some point, we will need to transform local parameters into a regular 3D transformation. Given some local coordinates $(\mathbf{m}_r, \alpha)$, we decompose the transformation into three independent steps, using an intermediate coordinate frame where we perform the rotation around $\alpha$:

$$T = T_{S\to G}^{-1} R_x(\alpha) T_{M\to G} \tag{5.13}$$

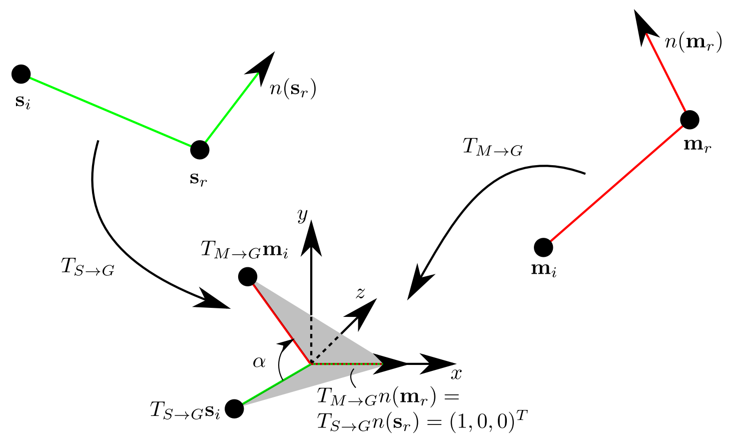


Figure 5.5: Building a rigid transformation that aligns two point pairs, $(\mathbf{m}_r, \mathbf{m}_i)$ and $(\mathbf{s}_r, \mathbf{s}_i)$, using an intermediate coordinate frame. $T_{M\to G}$ translates $\mathbf{m}_r$ into the origin and aligns the normal $n(\mathbf{m}_r)$ with the $x$-axis. $T_{S\to G}$ does the same for $\mathbf{s}_r$ and $n(\mathbf{s}_r)$, respectively. The two point pairs thus meet in the intermediate coordinate frame, where one degree of freedom remains, a rotation around the $x$-axis. The rotation $R_x(\alpha)$ aligns the transformed $T_{M\to G}\mathbf{m}_i$ and $T_{S\to G}\mathbf{s}_i$.

$R_x(\alpha)$ is the rotation around the $x$-axis with angle $\alpha$. $T_{M\to G} \in \mathrm{SE}(3)$ is defined as a transformation from model to intermediate coordinates that transforms $\mathbf{m}_r$ into the origin and its normal $n(\mathbf{m}_r)$ onto the $x$-axis:

$$T_{M\to G}\mathbf{m}_r = 0 \tag{5.14}$$
$$R(T_{M\to G})n(\mathbf{m}_r) = (1,0,0)^T \tag{5.15}$$

Sec. 2.1.3 discusses how such a transformation is constructed. Similarly, $T_{S\to G}$ does the same for $\mathbf{s}_r$:

$$T_{S\to G}\mathbf{s}_r = 0 \tag{5.16}$$
$$R(T_{S\to G})n(\mathbf{s}_r) = (1,0,0)^T \tag{5.17}$$

The transformation of an arbitrary model point $\mathbf{m}_i$ onto its scene point $\mathbf{s}_i$ is then given as

$$\mathbf{s}_i = T_{S\to G}^{-1} R_x(\alpha) T_{M\to G}\mathbf{m}_i \tag{5.18}$$

Fig. 5.5 illustrates this transformation.

**Discussion** Local parameters have several properties that allow them to be used in an efficient voting scheme. First, contrary to the translation component of SE(3), they are bounded. Second, they are highly regular and avoid the complex geometry of SO(3) by using only a 2D rotation, thus allowing a convenient regular parametrization. Compared to SE(3), which covers six degrees of freedom, local coordinates cover only three (two for the corresponding point, one for the rotation

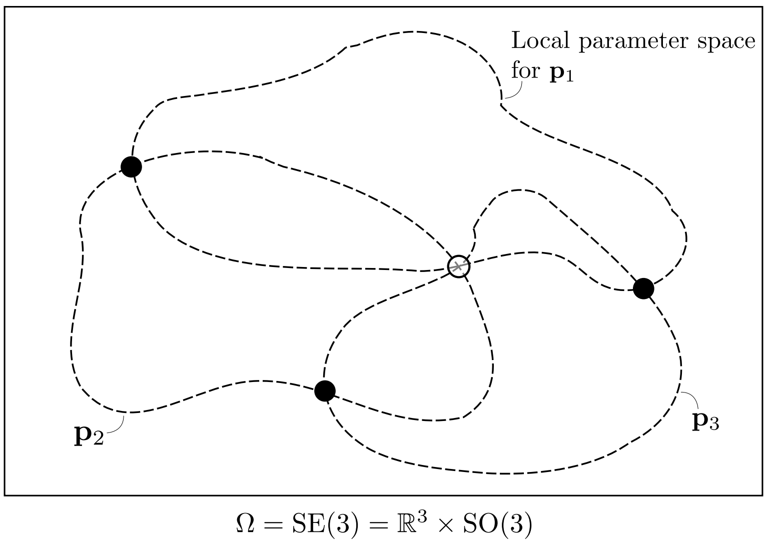

Local parameter space for $\mathbf{p}_1$
$\mathbf{p}_2$
$\mathbf{p}_3$
$\Omega = \mathrm{SE}(3) = \mathbb{R}^3 \times \mathrm{SO}(3)$

Figure 5.6: Illustration of the local parameter space. $\Omega = \mathrm{SE}(3)$ is the set of all rigid transformations. Each scene point (such as $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$) defines a manifold in $\Omega$ that contains only those transformations that align the model $M$ with the scene point. The transformations where manifolds intersect (dots) align two or more scene points with the model. We are interested in the transformation where the most manifolds intersect (white dot).

angle). Finally, there is a straightforward way of sampling local parameters regularly. Note that the local parametrization is *data-driven* in the sense that they are defined by the oriented input scene points.

As a downside, the described parameters are local to a single reference point in the scene, which is assumed to be on the object instance we search for. Since this information is in general not available beforehand, we will need to use several reference points throughout the scene, hoping that one or more lies on the target object. This adds two degrees of freedom, since those reference points are picked from a 2D manifold in 3D, the observed surface. In total, we will thus search through five degrees of freedom, instead of six for the general $\mathrm{SE}(3)$, by implicitly restricting our self to those transformations for which at least one transformed model point lies on the observed surface.

### 5.3.4 Voting Scheme

Given a fixed reference point $\mathbf{s}_r$ from the scene, we want to find the optimal local coordinates for which the number of points in the scene that are aligned with the model is maximized. Fig. 5.6 illustrates this. The optimization is done using a voting scheme, similar to the Generalized Hough transform. Once the optimal local coordinates are found, the global pose of the object can be recovered using (5.13).

For the voting scheme, a two-dimensional accumulator array is created that

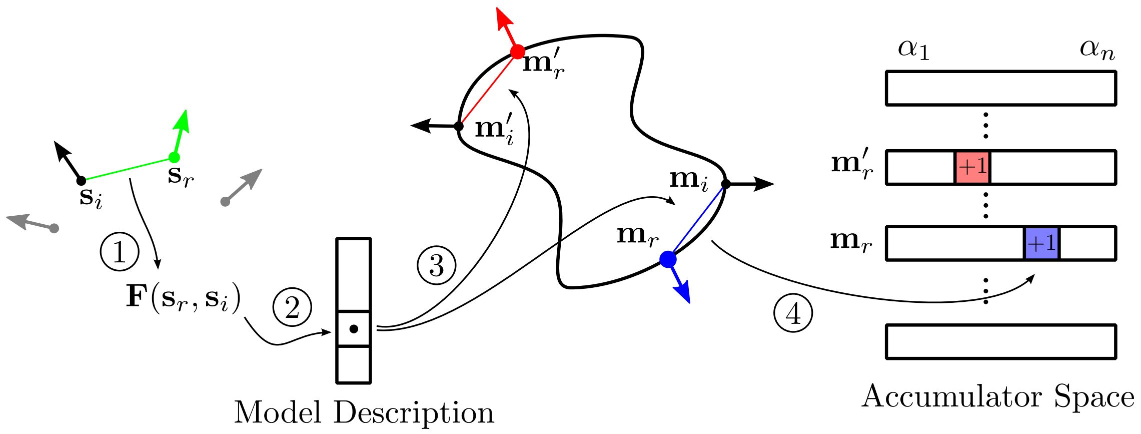


Figure 5.7: Visualisation of different steps in the voting scheme: (1) The reference point $\mathbf{s}_r$ is paired with every other point $\mathbf{s}_i$ and their point pair feature $\mathbf{F}$ is calculated. (2) $\mathbf{F}$ is matched to the global model description, which returns a set of point pairs on the model that have similar distance and orientation (3). For each point pair on the model matched to the point pair in the scene, the local coordinate $\alpha$ is calculated by solving (5.18). (4) After $\alpha$ is calculated, a vote is cast for the discretized local coordinate $(\mathbf{m}_r, \lfloor \alpha / n_{\text{angle}} \rfloor)$.

represents the local coordinate space $M \times [0; 2\pi[$. It has $|M|$ rows, one for each model point, and $n_{\text{angle}}$ columns, which is the number of discretization steps of the rotation angle $\alpha$. This accumulator array represents the discrete space of local coordinates for a fixed reference point.

For the actual voting, the reference point $\mathbf{s}_r$ is paired with every other point $\mathbf{s}_i \in S$ from the scene, and the model surface is searched for point pairs $(\mathbf{m}_r, \mathbf{m}_i)$ that have a similar distance and normal orientation as $(\mathbf{s}_r, \mathbf{s}_i)$. This search answers the question of where on the model the pair of scene points $(\mathbf{s}_r, \mathbf{s}_i)$ could be, and is performed using the pre-computed model description: The point pair feature $\mathbf{F}(\mathbf{s}_r, \mathbf{s}_i)$ is calculated and used as key to the hash table $H$ of the global model description, which returns the set of similar point pairs on the model.

For each match $(\mathbf{m}_r, \mathbf{m}_i)$, *i.e.*, for every possible position of $(\mathbf{s}_r, \mathbf{s}_i)$ on the model surface, the rotation angle $\alpha$ is calculated by solving (5.18) for $\alpha$ (compare Sec. 2.2.3). The transformation $T(\mathbf{m}_r, \alpha)$ then maps $(\mathbf{m}_r, \mathbf{m}_i)$ to $(\mathbf{s}_r, \mathbf{s}_i)$, as shown in Fig. 5.5. A vote is then cast for the discretized local coordinates $(\mathbf{m}_r, \lfloor \alpha / n_{\text{angle}} \rfloor)$. Fig. 5.7 and Fig. 5.8 outline the voting process.

After all points $\mathbf{s}_i$ are processed, the peak in the accumulator array corresponds to the optimal local coordinate, from which a global rigid movement can be calculated. For stability reasons, all peaks that received a certain amount of votes relative to the maximum peak are used as candidates in the subsequent processing. The number of votes of each bin correlates to the number of scene points that lie on the model if the model is transformed according to the corresponding local coordinates of the bin. The vote count is, however, affected by noise and by the binning problem of both the feature and the parameter discretization.

```
Input: Sampled model $M$
       Sampled scene $S$
       Reference point $\mathbf{s}_r \in S$

Initialize accumulator array of size $M \times n_{\text{angle}}$ with zeros
for $\mathbf{s}_i \in S$:
    Compute $\mathbf{f} = \hat{\mathbf{F}}(\mathbf{s}_r, \mathbf{s}_i)$
    Find $L = H(\mathbf{f})$
    for $(\mathbf{m}_1, \mathbf{m}_2) \in L$:
        Compute $\alpha$ using (5.18)
        Vote for $(\mathbf{m}_1, \lfloor \alpha / n_{\text{angle}} \rfloor)$

Find the peak $(m^*, \alpha^*)$ in the accumulator array
Compute its rigid transformation $T^*$ using (5.13)

Output: $T^*$
```

Figure 5.8: Basic voting scheme for a single reference point

### 5.3.5 Efficient Voting

Much of the time of the voting scheme itself is spent in the inner voting loop. To reduce the computational costs of the detection algorithm, we will now show how this inner voting loop can be implemented more efficiently. The challenges for this is to solve (5.18) quickly for all matching point pairs and to use a voting space representation that minimizes the work to be done per vote. While this section is more technical, it provides important speed-ups.

**Voting Angle Decomposition** To speed up solving (5.18) for every point pair in the list, we split $\alpha$ into two parts, $\alpha = \alpha_m - \alpha_s$, such that $\alpha_m$ and $\alpha_s$ depend only on the model and scene point pair, respectively. We split $R_{\mathbf{x}}(\alpha) = R_{\mathbf{x}}(-\alpha_s) R_{\mathbf{x}}(\alpha_m)$ and use $R_{\mathbf{x}}^{-1}(-\alpha_s) = R_{\mathbf{x}}(\alpha_s)$ to obtain

$$\begin{aligned} \mathbf{t} &= R_{\mathbf{x}}(\alpha_s) T_{S\to G} \mathbf{s}_i = \\ &= R_{\mathbf{x}}(\alpha_m) T_{M\to G} \mathbf{m}_i \in \mathbb{R}\mathbf{x} + \mathbb{R}_0^+ \mathbf{y}, \end{aligned} \tag{5.19}$$

*i.e.*, $\mathbf{t}$ lies on the half-plane defined by the $x$-axis and the non-negative part of the $y$-axis. Fig. 5.9 illustrates this decomposition.

For each point pair in the model or in the scene, a unique $\mathbf{t}$ exists. The angle between some $\mathbf{v}$ (for example, $\mathbf{v} = T_{S\to G}\mathbf{s}_i$) and $\mathbf{t}$ can be computed without explicitly computing $\mathbf{t}$, by projecting $\mathbf{v}$ into the $y$-$z$-plane and computing the

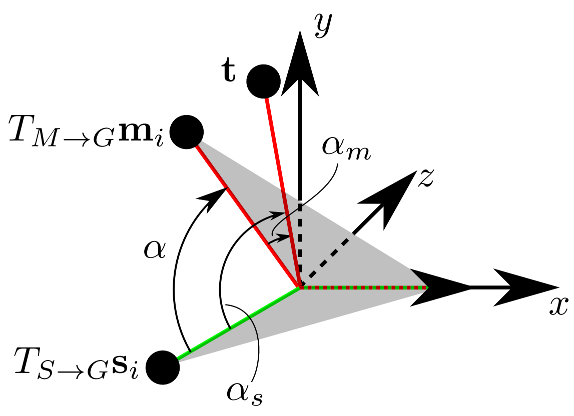


Figure 5.9: Decomposition of the rotation angle $\alpha$. After transforming a point pair $(\mathbf{s}_r, \mathbf{s}_i)$ into intermediate coordinates, $\mathbf{t}$ is the unique vector obtained by rotating $T_{S\to G}\mathbf{s}_i$ around the $x$-axis into the half-plane $\mathbb{R}\mathbf{x} + \mathbb{R}_0^+\mathbf{y}$. The vector $\mathbf{t}$ is uniquely defined, and the rotation angle $\alpha_s$ can be computed independently from other point pairs. For rotating one point pair onto another, their rotations $\alpha_s$ and $\alpha_m$ are computed. Then, $\alpha = \alpha_s - \alpha_m$.

angle to the positive $y$-axis using the atan2 function (see also Sec. 2.2.3):

$$\begin{aligned}\mathbf{v}' &= \mathbf{v} - \mathbf{e}_1(\mathbf{e}_1 \cdot \mathbf{v}) \\ \alpha &= \operatorname{atan2}(\mathbf{v}_y, \mathbf{v}_z)\end{aligned} \tag{5.20}$$

Because $\mathbf{t}$ is independent of the scene point pair, $\alpha_m$ can be precalculated for every model point pair in the offline phase and is stored in the model description. $\alpha_s$ needs to be calculated only once for every scene point pair $(\mathbf{s}_r, \mathbf{s}_i)$, and the final angle $\alpha$ is a simple difference of the two values. Note that the difference might be out of the range of $[0; 2\pi[$ thus needs to be normalized by computing the modulo w.r.t. $n_{\text{angle}}$.

Fig. 5.10 illustrates the offline stage and the voting for a single reference point.

**Angle Pre-Discretization** Another speed-up can be obtained by removing both the modulo operation and the integer casting from the inner voting loop. Note that both modulo and integer casting are among the slowest operations on CPUs that can additionally often not be pipelined.

To remove the integer casting from the inner voting loop, we cast the two angles $\alpha_s$ and $\alpha_m$ independently before the voting loop, and only add the two casted numbers:

$$\begin{aligned}\alpha_d &= \left\lfloor \frac{\alpha_m - \alpha_s}{n_{\text{angle}}} \right\rfloor \\ \Rightarrow \alpha_d &\approx \left\lfloor \frac{\alpha_m}{n_{\text{angle}}} \right\rfloor - \left\lfloor \frac{\alpha_s}{n_{\text{angle}}} \right\rfloor\end{aligned} \tag{5.21}$$

```
// Offline
Input: Sampled model $M$

for $\mathbf{m}_1, \mathbf{m}_2 \in M$ do:
  Compute $\alpha_m$ for $(\mathbf{m}_1, \mathbf{m}_2)$ using (5.19), (5.20)
  Store $(\mathbf{m}_1, \mathbf{m}_2, \alpha_m)$ in list $H(\hat{\mathbf{F}}(\mathbf{m}_1, \mathbf{m}_2))$

Output: Model description $H$

// Online
Input: Model description $H$
       Sampled scene $S$
       Reference point $\mathbf{s}_r \in S$

Initialize accumulator array
for $\mathbf{s}_i \in S$:
    Compute $\mathbf{f} = \hat{\mathbf{F}}(\mathbf{s}_r, \mathbf{s}_i)$
    Find $L = H(\mathbf{f})$
    Compute $\alpha_s$ for $(\mathbf{s}_r, \mathbf{s}_i)$
    for $(\mathbf{m}_1, \mathbf{m}_2, \alpha_m) \in L$:
        $\alpha = \alpha_m - \alpha_s$
        Sample and normalize: $\alpha_d = \lfloor \alpha / n_{\text{angle}} \rfloor \bmod n_{\text{angle}}$
        Vote for $(\mathbf{m}_1, \alpha_d)$

Find the peak $(m^*, \alpha^*)$ in the accumulator array
Compute its rigid transformation $T^*$ using (5.13)

Output: $T^*$
```

Figure 5.10: The offline and online phase, optimized by decomposing the local rotation angle $\alpha$ into a model and a scene-dependent component.

The cast of $\alpha_m$ can be performed in the offline stage, while the cast of $\alpha_s$ is performed once per scene point pair. Note that we only save casts if the voting list $L$ is larger than one. Note also that that (5.21) introduces the chance of an off-by-one error.[3] However, we found that that computational savings are worth this inaccuracy, and smoothing of the Hough space in angular direction mitigates this effect (see Sec. 5.3.6).

To remove the modulo operation from the voting loop, we use the fact that we know strict limits of the computed angle. Since $0 \leq \alpha_s, \alpha_m < 2\pi$, we know that

[3]For example, $\lfloor 1.6 + 1.6 \rfloor = \lfloor 3.2 \rfloor = 3$, but $\lfloor 1.6 \rfloor + \lfloor 1.6 \rfloor = 1 + 1 = 2$. The probability for such an error is 0.5.

$0 \leq \alpha_{s,d}, \alpha_{m,d} < n_{\text{angle}}$ and therefore

$$\begin{aligned} & -n_{\text{angle}} < \alpha_{m,d} - \alpha_{s,d} < n_{\text{angle}} \\ & \Rightarrow 0 < n_{\text{angle}} + \alpha_{m,d} - \alpha_{s,d} < 2n_{\text{angle}} \end{aligned} \tag{5.22}$$

Instead of normalizing $\alpha_d$ inside the voting loop, we introduce an ambiguity in the angular component of the voting space by re-sizing the voting space to $M \times \{0, 1, 2, \ldots 2n_{\text{angle}} - 1\}$.[4] The votes are then cast for $n_{\text{angle}} + \alpha_{m,d} - \alpha_{s,d}$ inside the voting loop, and the normalization is performed after voting by aggregating the votes of the cells $(\mathbf{m}, \alpha_d)$ and $(\mathbf{m}, \alpha_d + n_{\text{angle}})$.

The outer voting loop then reads

```
for $\mathbf{s}_i \in S$:
    Compute $\mathbf{f} = \hat{\mathbf{F}}(\mathbf{s}_r, \mathbf{s}_i)$
    Find $L = H(\mathbf{f})$
    Compute $\alpha_s$ for $(\mathbf{s}_r, \mathbf{s}_i)$
    $\alpha_{s,d} = \lfloor \alpha_s / n_{\text{angle}} \rfloor$
    for $(\mathbf{m}_1, \mathbf{m}_2, \alpha_{m,d}) \in L$:
        Vote for $(\mathbf{m}_1, n_{\text{angle}} + \alpha_{m,d} - \alpha_{s,d})$
```

**Pre-Compute Index** As a final implementation speedup, we can pre-compute the linearized voting index in the offline phase. The voting space $V$ is an array of size $n_{\text{angle}}|M|$, or $2n_{\text{angle}}|M|$ if the above angle pre-discretization is used. The array is linearized in memory in an angle-major order

$$\begin{aligned} V = [ & (\mathbf{m}_0, 0), (\mathbf{m}_0, 1), \ldots, (\mathbf{m}_0, 2n_{\text{angle}} - 1), \\ & (\mathbf{m}_1, 0), (\mathbf{m}_1, 1), \ldots, (\mathbf{m}_1, 2n_{\text{angle}} - 1), \\ & \ldots \\ & (\mathbf{m}_{|M|-1}, 0), (\mathbf{m}_{|M|-1}, 1), \ldots, (\mathbf{m}_{|M|-1}, 2n_{\text{angle}} - 1)] \end{aligned} \tag{5.23}$$

A vote for the local coordinates $(\mathbf{m}, \alpha_d)$ is thus cast to the linearized coordinates

$$\text{idx}(\mathbf{m}, \alpha_d) = 2n_{\text{angle}}\mathbf{m}_{\text{idx}} + \alpha_d \tag{5.24}$$

where $\mathbf{m}_{\text{idx}}$ is the index of $\mathbf{m}$ in $M$. Thanks to the above decomposition of $\alpha_d$ and the pre-discretization of the angles, we can decompose the linear index as

$$\begin{aligned} \text{idx}(\mathbf{m}, \alpha_d) &= 2n_{\text{angle}}\mathbf{m}_{\text{idx}} + n_{\text{angle}} + \alpha_{m,d} - \alpha_{s,d} \\ &= (2n_{\text{angle}}\mathbf{m}_{\text{idx}} + \alpha_{m,d}) + (n_{\text{angle}} - \alpha_{s,d}) \\ &= \text{idx}(\mathbf{m}, \alpha_{m,d}) + (n_{\text{angle}} - \alpha_{s,d}) \end{aligned} \tag{5.25}$$

[4] Note that the angular component could start at 1 according to (5.22). However, analysis of the voting space is more elegant if we simply double the angular component.

```
// Offline
Input: Sampled model $M$

for $\mathbf{m}_1, \mathbf{m}_2 \in M$ do:
  Compute $\alpha_m$ for $(\mathbf{m}_1, \mathbf{m}_2)$ using (5.19), (5.20)
  $\alpha_{m,d} = \lfloor \alpha_m / n_{\text{angle}} \rfloor$
  $idx_m = 2n_{\text{angle}}\mathbf{m}_{1,\text{idx}} + \alpha_{m,d}$
  Store $idx_m$ in $H_d(\hat{\mathbf{F}}(\mathbf{m}_1, \mathbf{m}_2))$

Output: Model description $H$
```

Figure 5.11: The offline phase of the voting scheme after including several optimizations.

Note that $\text{idx}(\mathbf{m}, \alpha_{m,d})$ is scene-independent. We can thus pre-compute the linearized indices for each list $L = H(\mathbf{f})$ in the model generation phase:

$$L_d = \{\text{idx}(\mathbf{m}, \alpha_{m,x}) : (\mathbf{m}, \alpha_{m,x}) \in L\} \tag{5.26}$$

The pre-linearized lists are stored in the hash table $H_d$. This leads to the voting loop

```
Lookup $L_d = H_d(\hat{\mathbf{F}}(\mathbf{s}_r, \mathbf{s}_i))$
for $idx \in L_d$ do:
  $V[idx + n_{\text{angle}} - \alpha_{s,d}]{+}{+}$
```

By adjusting the pointer to the voting space before the loop, we finally arrive at the elegant and tight voting loop

```
Lookup $L_d = H_d(\hat{\mathbf{F}}(\mathbf{s}_r, \mathbf{s}_i))$
$V_{\text{local}} = V + n_{\text{angle}} - \alpha_{s,d}$
for $idx \in L_d$ do:
  $V_{\text{local}}[idx]{+}{+}$
```

Fig. 5.11 summarizes the offline and Fig. 5.12 online phase of the voting scheme using the above modifications.

**Duplicate Elimination** From each list $L_d$ of pre-computed voting indices, we remove duplicates (thus effectively making the list a set). This aids two things: First, it avoids that over-sampling the original model leads to larger lists. Instead, even if two or more point pairs lead to the same feature and the same voting space index, only a single one is retained. Because of this, the lists will converge for a infinitely densely sampled model. It is thus impossible to overtrain the model.

```
// Online
Input: Model description $H$
       Sampled scene $S$
       Reference point $\mathbf{s}_r \in S$

Initialize accumulator array: $V \leftarrow 0_{|M|\, 2n_{\text{angle}}}$
for $\mathbf{s}_i \in S$:
    Compute $\mathbf{f} = \hat{\mathbf{F}}(\mathbf{s}_r, \mathbf{s}_i)$
    Find $L_d = H_d(f)$
    Compute $\alpha_s$ for $(\mathbf{s}_r, \mathbf{s}_i)$
    $\alpha_{s,d} = \lfloor \alpha_s / n_{\text{angle}} \rfloor$
    $V_{\text{local}} = V + n_{\text{angle}} - \alpha_{s,d}$
    for $idx \in L_d$ do:
        $V_{local}[idx]{+}{+}$

// Aggregate ambiguous votes
for $\mathbf{m} \in M$, $\alpha \in \{0, 1, \dots n_{\text{angle}} - 1\}$ do:
    $V'[(\mathbf{m}, \alpha)] = V[(\mathbf{m}, \alpha)] + V[(\mathbf{m}, \alpha + n_{\text{angle}})]$

Find the peak $(m^*, \alpha^*)$ in the accumulator array
Compute its rigid transformation $T^*$ using (5.13)

Output: $T^*$
```

Figure 5.12: The online phase of the voting scheme after including several optimizations. Note the tight inner voting loop which is stripped of all expensive operations.

Instead, only the sampling of the feature and of the Hough space influences the list lengths.

Second, removing duplicates allows a direct interpretation of the voting result: The number of votes for a pose is the number of scene points that would lie on the object, if transformed using that pose (times a factor for the probability of matching point pairs correctly).

### 5.3.6 Analysis of the Voting Space

**Binning Problem** One concern when using the Hough transform is that the sampling of the parameter space leads to non-smooth distributions of votes: A vote might fall close to the borders between two samples, and noise in the observed features makes it fall into either one of both. This effect is sometimes called the *binning problem*. More noise in the observed features intensifies this effect, since that noise is often transformed into noise in the recovered parameter. The effect also increases with the number of dimensions of the Hough space, since the probability increases that a vote lies close to the boundary of a parameter

```
Input: Voting Space $V$
       Gaussian smoothing $\sigma$
       Threshold $\delta_v$

for $v$ in $V$ do:
  $V_{\text{neigh}}(v) \leftarrow \{\}$
  for $u$ in $V$ do:
    Compute $w_{u,v}$ using (5.28)
    if $w_{u,v} > \delta_v$ do:
      $V_{\text{neigh}}(v) \leftarrow V_{\text{neigh}}(v) \cup \{u\}$

Output: Neighborhoods $V_{\text{neigh}}$ and weights $w$
```

Figure 5.13: Computation of the neighborhoods within the voting space. Each entry in the voting space corresponds to a pose, and this algorithm computes the similarity between those poses.

space sample.

The binning problem leads to several undesired effects: First, the peak might have fewer votes than it should, since some votes are cast to neighboring bins. The votes are effectively smoothed over a neighborhood of bins. Second, local maxima might arise around the global maximum that should actually be part of the global maximum. Third, it forces a trade-off regarding the sampling size of the parameter space, *i.e.*, the sizes of the bins: Larger bins reduce the effect of the binning problem, while smaller bins allow for a more precise recovery of the parameters.

Three basic strategies can be employed to counter this effect:

1. **Rely on statistics** (in other words, **simply do nothing, but a lot of it**). If enough votes are cast, the law of large numbers will eventually lead to a smoothing of the votes such that the correct bin receives the most votes. However, this technique can be rather unreliable, is potentially more expensive than other techniques – in particular if the peak in the voting space would be available with fewer votes, and additional votes are cast only to perform this smoothing – and does not solve the problem of neighboring local maxima.

2. **Smooth while voting.** Multiple, potentially weighted, votes can be cast to the surrounding bins: By casting votes using a small smoothing kernel, the binning problem can be avoided. This technique makes the *voting step more expensive*, since multiple votes must be cast instead of just one. Additionally, it might require continuous vote counters instead of integer-based counters.

The size of the smoothing kernel can be adjusted based on the expected noise in the observed features.

3. **Smooth after voting.** The parameter space can be smoothed after voting, using some smoothing kernel that incorporates the similarity between neighboring parameters. The size of the smoothing kernel can again be adjusted based on the noise in the observations. This technique makes the *peak extraction more expensive*.

In general both the smooth-while-voting and smooth-after-voting are equivalent on a functional level: If the same smoothing kernel is used, both techniques will result in the same number of votes per bin. In terms of performance, smooth-while-voting incurs costs per vote, while smooth-after-voting incurs costs per bin. Which one is faster thus depends on the number of expected votes vs. the size of the voting space, something that is difficult to estimate beforehand, since it might be scene-dependent.

**Our Approach** In our approach, we use a mixture of the statistical and the smooth-after-voting technique that avoids most of the performance costs while still avoiding the binning problem. We assume that after voting, the dominant peaks will have enough votes to be detectable in a non-smoothed voting space. We then smooth the voting space only in a small neighborhood around those detected maxima. This combines the advantage of speed of the statistical approach with the accuracy of the smoothing approaches.

**Smoothing Kernel** In order to smooth the voting space, we need to define some smoothing kernel. While our parameter space has a certain regularity in the dimension of the rotation angle, it is more complex in the dimension of the corresponding model point. We compare the similarity of parameters based on the similarity of their corresponding rigid 3D transformations, based on the single-valued, model-dependent $m_{1,\text{norm}}$ metric (see Sec. 2.2.4, (2.37)).

Let $T_v \in \text{SE}(3)$ be the corresponding 3D transformation of a bin $v \in V$ of the voting space, and let $c(v)$ be the number of received votes, we define the smoothed variant $\hat{c}(v)$ as

$$\begin{aligned}\hat{c}(v) &= \frac{1}{a} \sum_{u \in V} w_{u,v} c(u) \\ &= \frac{1}{a} \sum_{u \in V} w(m_{1,\text{norm}}(T_u, T_v)) c(u) \qquad (5.27)\end{aligned}$$

where $a = \sum_{u \in V} w_{u,v}$ is a normalization factor. Several smoothing kernels $w$ are possible, we use a classic Gaussian kernel

$$w(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \qquad (5.28)$$

Since the parameter to the Gaussian function, $m_{1,\text{norm}}$, is normalized, we can use a fixed, model-independent value of $\sigma = 0.03$ (*i.e.*, 3% of the model's diameter).

**Precomputing Weights** Note that the smoothing weights $w_{u,v}$ can be computed offline, since they are fixed for a given model. The weights are also close to zero for most combinations of $u$, $v$, such that we can limit the sum in (5.27) to a few neighbors only. This allows for an efficient online smoothing that evaluates

$$\hat{c}(v) = \frac{1}{a} \sum_{u \in V_{\text{neigh}}(v)} w_{u,v} c(u) \tag{5.29}$$

where $V_{\text{neigh}}(v)$ is the (small) set of neighboring voting space cells of $v$,

$$V_{\text{neigh}}(v) = \{w \in V : w_{u,v} > \delta_v\} \tag{5.30}$$

In practice, we set $\delta_v = 0.01$. As example, for a model sampled with 3% of its diameter, an angular sampling of the Hough space in steps of 12°, and $\sigma = 0.03$, average neighborhood sizes are $|V_{\text{neigh}}| \approx 5$. Fig. 5.13 summarizes the offline phase for computing the neighborhoods.

**Sub-Bin-Precise Peak Extraction** As mentioned above, the bin sizes of the voting space are selected as a trade-of between accuracy and performance: Smaller bins are more accurate, while larger bins are typically faster and less prone to noise. Instead of making the bins smaller, accuracy can also be improved by extracting the peak with sub-bin-precision (see, for example, [148]): After finding the maximum of the smoothed voting space,

$$v^* = \arg\max_{v \in V} \hat{c}(v) \tag{5.31}$$

we can use the votes of neighboring bins to estimate the error due to binning. Intuitively, if the bins in one direction (in SE(3)) have significantly more votes than in the other direction, the real maximum lies more in that direction. Formally, we can fit a Gaussian or a quadratic function into the local neighborhood of the peak, and use the maximum of that function as the peak position. The function to fit depends on the expected spread function of the votes, which is difficult to obtain.

In our case of SE$(3)$, however, fitting a quadratic (or Gaussian) function might be difficult, since the pose samples surrounding the peak all lie on the manifold determined by the local coordinates (compare Sec. 5.3.3). The fitted function is thus also valid only on that manifold. We thus use a different approach: Instead of using a function fit, we compute the weighted average $\overline{v}$ over the neighboring poses (see, for example, [95, 59]):

$$\overline{T}_v = \frac{1}{a} \sum_{u \in V_{\text{neigh}}(v)} w_{u,v} T_u \tag{5.32}$$

```
Input: Set of poses and scores $(T,s) \in P \subset \mathrm{SE}(3) \times \mathbb{R}^+$
       Metric $d$, Clustering threshold $d_{max}$
       Minimum score $s_{min}$

$P^* = \{\}$
while $\max_{s \in S} s > s_{min}$ do:
  Find peak pose: $p^* = (T^*, s^*) = \arg\max_{(T,s) \in P} s$
  Find surrounding cluster: $N(p^*) = \{p \in P : d(p, p^*) < d_{max}\}$
  Remove cluster from remaining set: $P \leftarrow P \setminus N(p^*)$
  Compute weighted mean: $W(N(p^*))$
  Add to output set: $P^* \leftarrow P^* \cup W(N(p^*))$

Output: Set of smoothed, filtered poses $P^*$
```

Figure 5.14: Algorithm for pose clustering. We use the $m_{1,\mathrm{norm}}$-metric from Sec. 2.2.4 as pose comparison metric $d$.

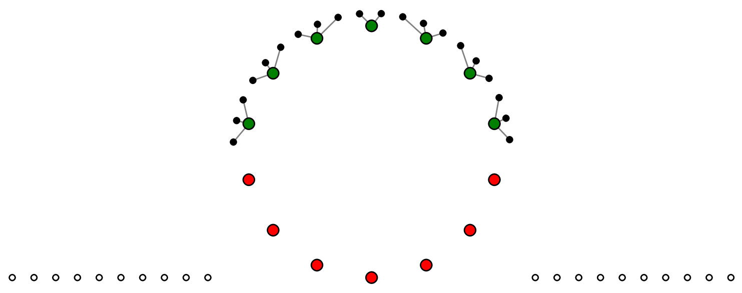
We compared the accuracy of the rigid transformations obtained with and without sub-bin precision on a synthetic scene with known ground truth. Overall, the sub-bin precise peak extraction improved the accuracy of the extracted transformation by around 0.4% of the model's diameter w.r.t. the $m_{1,\mathrm{norm}}$ metric.

### 5.3.7 Pose Clustering

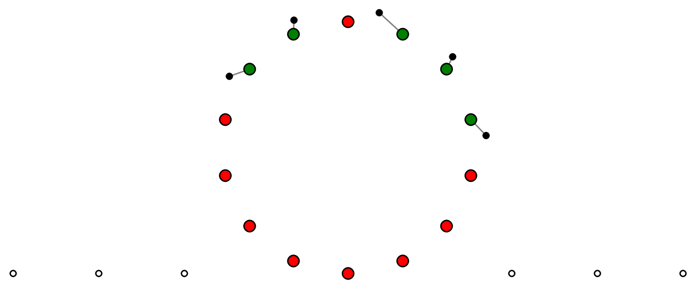
The above local voting scheme identifies the object pose if the reference point lies on the surface of the object. Multiple reference points are necessary to ensure that at least one of them lies on the target object instance. As shown in the previous section, each reference point returns a set of possible object poses that correspond to peaks in its accumulator array. The retrieved poses will only approximate the correct pose owing to different sampling rates of the scene and the model and the sampling of the rotation in the local coordinates. We now introduce an additional step that filters out incorrect poses, removes duplicates and increases the accuracy of the poses.

To this end, the poses computed in the voting step are clustered in a greedy fashion. We first find the pose with the highest voting score, and then assign all other poses that are close w.r.t. $m_{1,\mathrm{norm}}$ (see Sec. 2.2.4) to its cluster. The score of a cluster is the maximum of the scores of the contained poses. This step is effectively a non-maximum-suppression in the space of rigid transformations.

After finding the cluster with the maximum score, the resulting pose is calculated by computing the weighted average of the poses contained in the cluster, thereby increasing the accuracy of the pose. Since multiple instances of the object can be in the scene, several clusters can be returned by the method by



(a)



(b)

Figure 5.15: (a) Illustration of the scoring function. Imagine the scene (small points) contains a ball on a plane, and is seen from above. The large points are points on the registered model. For each scene point, we find the closest model point and mark it as visible (green). All other model points are invisible (red). (b) Illustration of the sampling problem when computing the score. If the scene is less dense than the model, some model points will incorrectly be marked as invisible, even though they are visible.

removing all poses from one cluster before continuing. Pose clustering increases the stability of the algorithm by removing isolated poses with low scores, and the averaging step increases the accuracy of the final pose. Fig. 5.14 illustrates the greedy clustering process.

### 5.3.8 Efficient Scoring

For many applications, it is important to have some knowledge about the quality, or a *score*, of the detected object instances. For example, robots can often pick up only the topmost object of a pile of objects, and would want to select the object that is most visible. Also, if no instance of the object is present in the scene, the proposed method might still return a false positive by aligning a small part of the object with background clutter. A robust score can help to distinguish between such false positives and true positive matches.

```
Input: Sampled model point cloud $M$
       Pre-computed nearest neighbor structure NN for $M$
       Scene point cloud $S$
       Distance threshold $\epsilon$
       Object pose $T$

for $\mathbf{m} \in M$ set $\text{present}[\mathbf{m}] \leftarrow \text{FALSE}$

for $\mathbf{s} \in S$ do:
    $\mathbf{m} \leftarrow \text{NN}(T^{-1}\mathbf{s}, M)$
    if $|\mathbf{s} - \mathbf{m}| < \epsilon$
        $\text{present}[\mathbf{m}] \leftarrow \text{TRUE}$

$n_{\text{present}} = |\{\mathbf{m} \in M : \text{present}[\mathbf{m}] = \text{TRUE}\}|$
$s = n_{\text{present}} / |M|$

Output: Normalized score $s$
```

Figure 5.16: Algorithm for computing the normalized score that represents the ratio of visible scene surface.

A good score should enable two things. First, it should allow to order the list of detected instances, and as such provide a *relative* measurement for comparing detection results. Second, it should provide an *absolute* measurement that helps distinguish false positives from true positives.

In terms of usability, scores should be normalized, usually to the range $[0, 1]$ where 0 represents a particularly bad result and 1 a very good result. This allows users to set score thresholds that are independent of model and scene. It also helps if scores have some simple, physical interpretation, as this allows users to get an intuition of the quality of the results. Finally, scores should be robust against minor distortions such as small amounts of noise or a different sampling of the scene's surface.

**Related Work** The problem of scoring the results can also be seen to be a hypothesis verification. Relatively little work has been done in this regard [2]. Most recently, Aldoma *et al.* [2] proposed a global approach for hypothesis verification that works in heavily cluttered environments and shows remarkably good results on standard datasets. However, they report verification times of $\approx 4s$ per scene, which is too long for many practical applications. Contrary to their approach, our scoring or verification is local in nature, works significantly faster and requires only few parameters, which can usually be left unadjusted.

**Voting Score** A first and straightforward score would be the number of votes that an instance received. Since this score is a natural by-product of the proposed method, it comes without additional costs. It also has a physical interpretation, namely that it represents the number of scene points that would lie on the object instance.

However, several reasons speak against the number of votes as score. First, it is non-normalized and changes with several parameters (such as the scene sampling, feature sampling, and the number of reference points). Second, and more importantly, it is not robust enough, as it is a statistical measurement. Each vote is cast to the correct bin only with a certain probability, which is affected by noise, feature sampling, and voting space sampling (compare Fig. 5.3). We performed experiments that show that a simple rigid transformation of the scene can change the number of votes for a pose by up to 25%.

**Visible Surface Ratio** A more robust score that is the ratio of the visible object surface vs. the total object surface. For example, if half of the object is visible, the score should be 0.5. This score is normalized, has an intuitive interpretation, and is – if computed after the pose refinement – highly robust. It does, however, require a parameter that defines how far a scene point may be away from the object's surface to still be interpreted to be "on" the surface. This parameter should be in the range of the expected noise of the measured points.

To efficiently compute this score, we sample the model and use the nearest neighbor structure from Sec. 3 to find the closest model point for each scene point. Each sampled model point is classified as *present* if some scene point is close to it, and as *absent* if not. The ratio of present to total scene point defines the score. Fig. 5.16 outlines the algorithm and Fig. 5.15a illustrates the scoring process.

Note that an assumption of this scoring is that the scene points are more dense than the sampled model points. If this assumption is not met – for example, if the sensor has a very low resolution, or the object instance is very far from the sensor –, the scoring will produce an incorrectly low score, since some of the actually visible model points will be classified as invisible (see Fig. 5.15b).

In practice, when using a single 3D sensor, only one side an object instance is visible. If a CAD model is used that contains all sides of an object, the score is often below 0.5 even for perfect matches. Note that if the calibration of the original sensor is available, the visibility of the object given the found position can be used to compute the score as ratio of ratio of the visible object surface vs. the potentially visible object surface can be computed and used as score. We did not do this for two reasons: First, the method is designed to be independent of the originally used sensor and works even if the calibration data is not available. And second, it introduces a bias towards poses where only a small part of the object is potentially visible (such as seeing a long cylinder exactly from top or bottom, such that only the cap is visible). Of course, the score we use has a bias

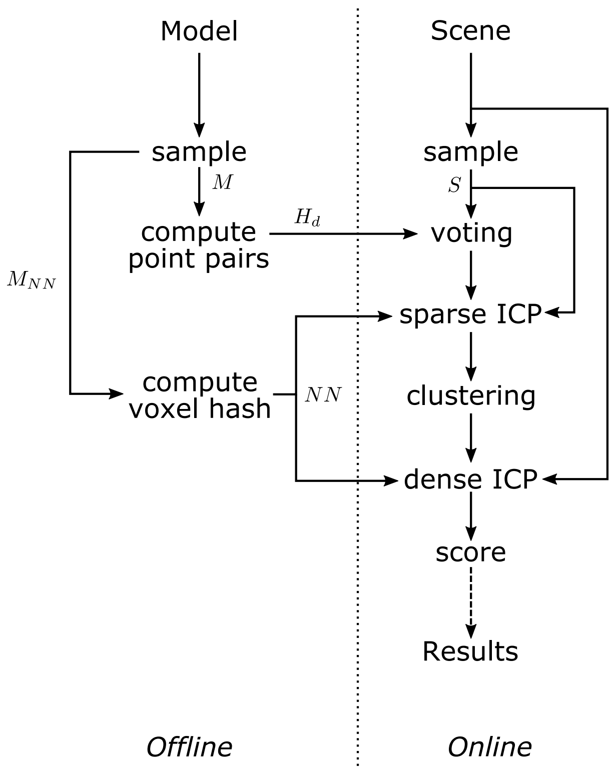


Figure 5.17: The different steps of the detection pipeline. In the offline phase (left), the model is sampled twice: Once for the computation of the point pair features, and once for the nearest neighbor data structure. In the online phase (right), the scene is sampled and the local voting is used to find pose candidates. The candidates are first approximately refined, using the sampled scene. The clustering step effectively filters out the best poses and performs non-maximum suppression. The remaining poses are then refined more accurately, using the complete scene. Finally, the score is computed.

towards

### 5.3.9 Detection Pipeline

To complete the description of the proposed method, Fig. 5.17 outlines the complete detection pipeline. In the following, each step is briefly summarized.

#### Offline Phase

- **sampling**: The model is first sampled twice, once for the computation of the point pair features, and once for the creation of the nearest neighbor lookup database (Sec. 2.3)
- **point pair computation**: All point pairs are formed, and their linearized voting indices are stored in the hash table $H_d$, using the sampled point pair

features as key (Sec. 5.3.2 and Sec. 5.3.5)

- **voxel hash computation**: The voxel-hash based nearest neighbor structure is created (Sec. 3)

**Online Phase**

- **sampling**: The scene is first sampled to avoid bias towards more densely sampled parts of the scene, as well as for performance reasons
- **voting**: Reference points are sampled uniformly from the scene, and the voting in the corresponding local parameter spaces is used to find pose candidates (Sec. 5.3.4)
- **sparse ICP**: A rapid ICP is performed, using the sparsely sampled scene (Sec. 4, especially Sec. 4.5 for a motivation of the dense and sparse ICP)
- **clustering**: The poses are clustered in pose space. This step selects the locally best poses and effectively performs a non-maximum-suppression (Sec. 5.3.7)
- **dense ICP**: The selected poses are accurately refined using all available scene points.
- **scoring**: A score is computed for each pose that represents the quality of the pose. The poses are sorted based on the score (Sec. 5.3.8)

### 5.3.10 Complexity Analysis

This section discusses the theoretical computational costs of the proposed voting scheme. The local coordinates $(\mathbf{m}_r, \alpha)$ describe three degrees of freedom: two for $\mathbf{m}_r$, which is sampled from the 2D surface of the model $M$, and one for the angle $\alpha$. The reference points are sampled from the 3D surface of the scene, which is a 2D manifold, therefore representing two degrees of freedom. Overall, the voting thus examines five degrees of freedom, compared to six degrees of freedom for a generic rigid transformation in SE(3). The remaining degree of freedom is reduced by the condition that the model must be stuck to the scene surface, thus avoiding positions where the model would float in empty space.

Since we sample the scene uniformly, for each reference point, the number of neighboring points that are closer to the reference point than the diameter of the object is bounded. The lengths of the voting lists, $|H(\mathbf{f})|$, are also bounded. The costs for voting for a particular reference point is thus constant, $\mathcal{O}(1)$. Since the reference points are also sampled uniformly from the scene, and since the sampling rates are based on the diameter of the model, the total runtime is

$$\mathcal{O}\left(\frac{\operatorname{diam}(S)}{\operatorname{diam}(M)}\right) \tag{5.33}$$

In other words, the runtime of the voting scheme is linear in the size of the scene compared to the size of the model. Note, however, that the constant factor is dominated by the list lengths $|H(\mathbf{f})|$, as shown in Sec. 5.4.3.

## 5.4 Experiments

We evaluated our method against a large number of synthetic and real datasets and tested the performance, efficiency, and parameter dependence of the algorithm.

For all experiments, the feature space was sampled by setting $d_{\text{dist}}$ to be relative to the model diameter $d_{\text{dist}} = \tau_d \ \text{diam}(M)$. Unless stated otherwise, the sampling rate $\tau_d$ was set to 0.05. This makes the parameter independent from the model size. The normal orientation was sampled with $n_{\text{angle}} = 30$. This allows a variation of the normal orientation with respect to the correct normal orientation of up to $12^\circ$. Both the model and the scene point clouds were subsampled such that all points have a minimum distance of $d_{\text{dist}}$. One fifth of the points in the subsampled scene were used as reference points. After resampling the point cloud, the normals were recalculated by fitting a plane into the neighborhood of each point. This step ensures that the normals correspond to the sampling level and avoids problems with fine details, such as wrinkles on the surface. In other words, it avoids aliasing problems with the normals. Both the scene and the model were sampled in the same way. The same parameters were used for all experiments except where noted otherwise.

We will show that our algorithm has a superior performance and allows an easy trade-off between speed and recognition rate. The algorithm was implemented in C and parallelized. The benchmarks were run on a 3.20 GHz Intel Core i5-4570 with 16 GB RAM. All given timings measure the whole matching process including the scene subsampling, normal calculation, voting, and pose clustering. The individual experiments note whether the refinement and scoring steps were included in the timings. For the construction of the global model description, all point pairs in the subsampled model cloud were used. The construction took at most several seconds for each model.

### 5.4.1 Synthetic Data

**Voting** We first evaluated the proposed voting scheme against a large set of synthetically generated three-dimensional scenes. Synthetic data has the advantage that we know the exact ground truth and have tight control over noise, clutter and occlusion. We selected the four models shown in Fig. 5.18, the T-Rex, the Chef, the Bunny and the Clamp, to generate the synthetic scenes. The chosen models demonstrate different geometries.

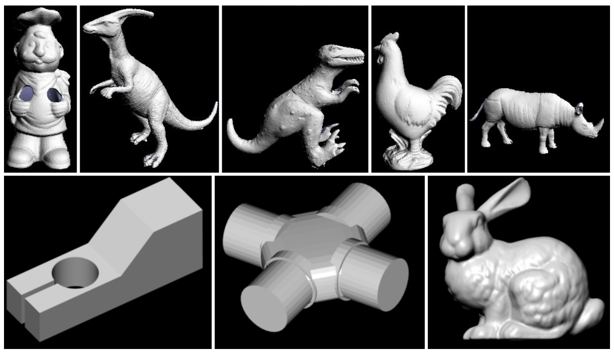


Figure 5.18: Models used in the experiments. Top row: The five objects from the scenes of Mian *et al*. [93, 92]. Note that the rhino was excluded from the comparisons, as in the original paper. Bottom row from left to right: The Clamp and the Cross Shaft from MVTec, and the Stanford Bunny from [124].

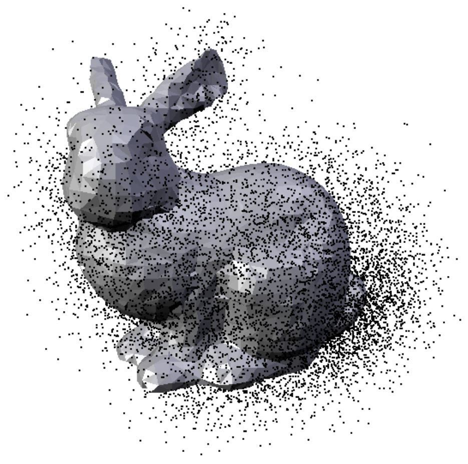
In the first set of experiments, scenes containing only a single object were used, and the performance with respect to **noise** was measured. Each of the four afore-mentioned objects was rendered from 50 different directions. This results in 200 point clouds, which were then corrupted by additive Gaussian noise with a standard derivation given relative to the object's diameter. This was done prior to the subsampling step. We were interested in the recognition rate, *i.e.*, the number of scenes where the object was successfully found. An object is defined to be detected if the error of the resulting pose relative to the ground truth is smaller then some predefined threshold. In our experiments the threshold was set as $m_{1,\text{norm}} < 0.1$ (see (2.37)). Fig. 5.19 shows an example scene and the recognition rates.

In a second set of experiments, 50 artificial scenes were rendered, each containing from four to nine randomly placed objects from the four objects used above. The scenes thus show **multiple instances**, **clutter**, and **occlusion**. In total, 347 objects were placed in 50 scenes. We measured the performance of our algorithm with respect to occlusions and in case of real data also with respect to clutter. The definition of [75] for occlusion and [92] for clutter, both defined per object instance, were used:

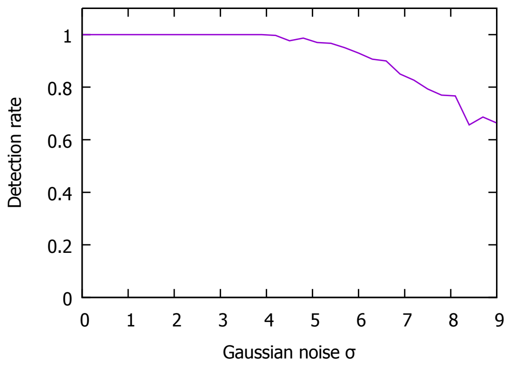
$$\text{occlusion} = 1 - \frac{\text{model surface area in the scene}}{\text{total model surface area}}, \tag{5.34}$$

$$\text{clutter} = 1 - \frac{\text{model surface area in the scene}}{\text{total surface area of scene}}. \tag{5.35}$$

The average number of points in the subsampled scenes is $|S| \approx 1690$. We ran out algorithm three times, using 1/5th, 1/10th and 1/40th of the scene points as reference points. Fig. 5.20 shows an example scene and the recognition rates. Both the recognition rate and the execution time depend on the number of used

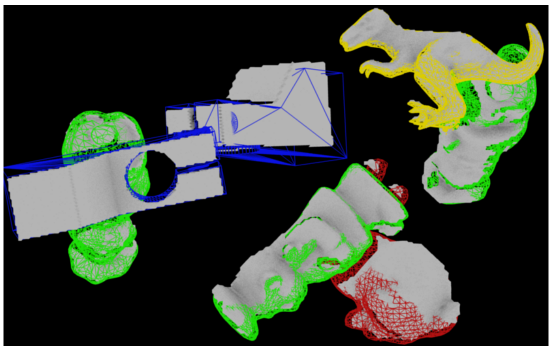


(a)

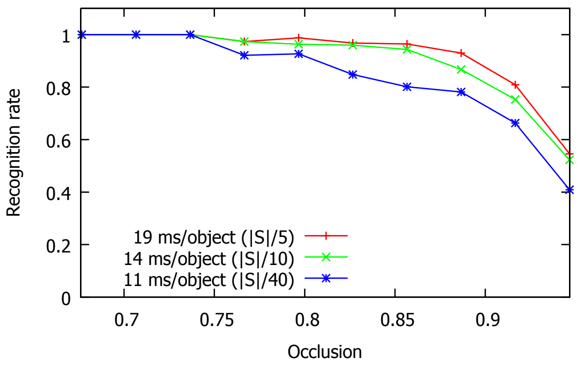


(b)

Figure 5.19: Results for the artificial scenes with a single object. (a) A point cloud with additive Gaussian noise added to every point ($\sigma = 5\%$ of the model diameter). The pose of the bunny was recovered in $\approx$ 5ms. (b) Detection rate for different levels of Gaussian noise, given as standard deviation in percent of the object's diameter. For each noise level, the detection rate is averaged over 300 scenes.

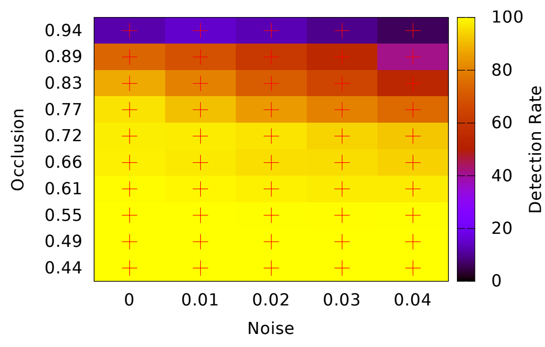


(a)

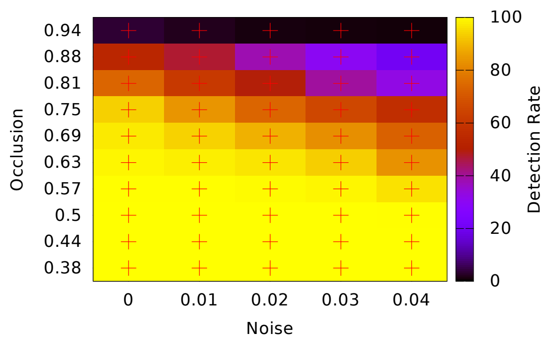


(b)

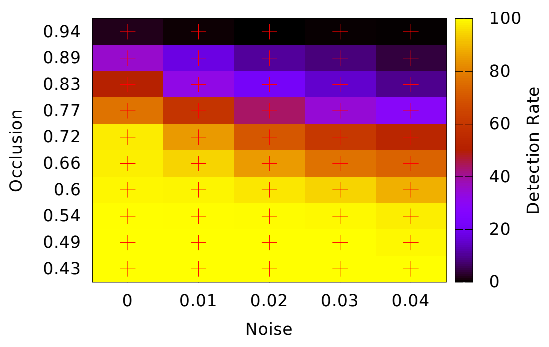
Figure 5.20: (a) One of the 50 synthetic scenes with the detected objects overlayed as colored wireframe. The poses are the result of the voting only, without any pose refinement such as ICP. (b) Recognition rate against occlusion for the synthetic scenes with multiple objects. The number of reference points was $|S|/5$, $|S|/10$ and $|S|/40$ respectively.



(a) Pipe Joint

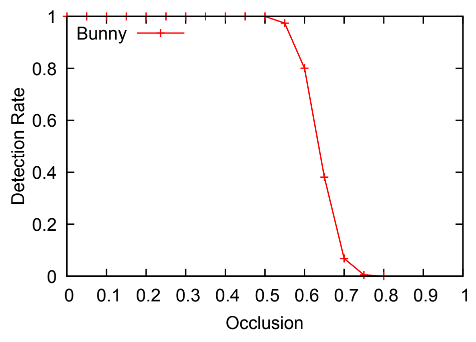


(b) Bunny

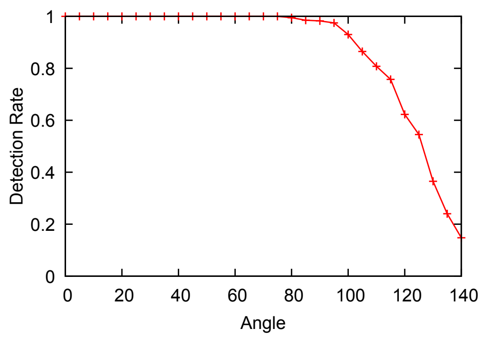


(c) Connection Rod

Figure 5.21: Effect of occlusion and noise on the detection pipeline. Several 10.000 scenes were rendered, each containing multiple, partially occluded, instances of all three objects, as well as Gaussian noise. The resulting detection rates are plotted w.r.t. the relative noise $\sigma_{rel}$ as well as the occlusion. Note that the bottom side of the bunny is not fully modeled, allowing for occlusions of less than 50%.



(a)



(b)

Figure 5.22: Results where the object was rotated w.r.t. the reference model. The reference model was created by rendering the object in a random orientation, using only the then visible points. For detection, the object was again put into random orientations and rendered. We measured how much of the reference model was still visible (a) and by how much the object was rotated w.r.t. its reference pose ((b), excluding in-plane rotation) vs. the detection rate. Overall, Occlusions up to 50% and rotations up to $\approx 80^\circ$ are are handled perfectly.

reference points. For $|S|/5$ reference points, 89.3% of all objects were correctly detected. The missed objects are mostly the highly occluded ones: 98% of all objects with more than 15% of visible surface, *i.e.*, less than 85% occlusion, were found. For $|S|/40$ reference points, 77.2% of all objects and 89.1% of objects more than 15% visible were found. However, the latter setup was roughly twice as fast.

From the experiments it is obvious that the number of reference points is a tradeoff between speed and performance: A significantly faster matching can be achieved at the price of not detecting ill-conditioned objects with high occlusion.

**Noise vs. Occlusion** Another test was performed to evaluate the effects of simultaneous occlusion and Gaussian noise on the detection performance. Fig. 5.21 shows the cumulative results. Overall, the detection rate stays high for moderate occlusions and noise. The runtime per detected object was around $100ms$. As explained above, a denser sampling would have led to a more accurate detection, at the cost of runtime. Note also that in practice, some of the Gaussian noise could usually be filtered prior to detection by smoothing the input data. The noise levels shown in the experiments are thus rather extreme and can usually be avoided in real-world setups.

**Rotation of Model vs. Scene** An additional set of synthetic tests evaluated by how much an object can be rotated and still be detected. We first rendered an object from a random direction and created a detection model from the rendered scene. The model was then rendered from different random directions (test scenes) and detected using that model. We first measured the detection rate w.r.t. the model's rotation angle between model and scene. Only the out-of-plane angel is measured, *i.e.*, the in-plane rotation of the object around the camera's $z$-axis was ignored, since the method is invariant against such rotation. A second test measured the detection rate w.r.t. how much of the surface in the model scene was still visible in the test scene.

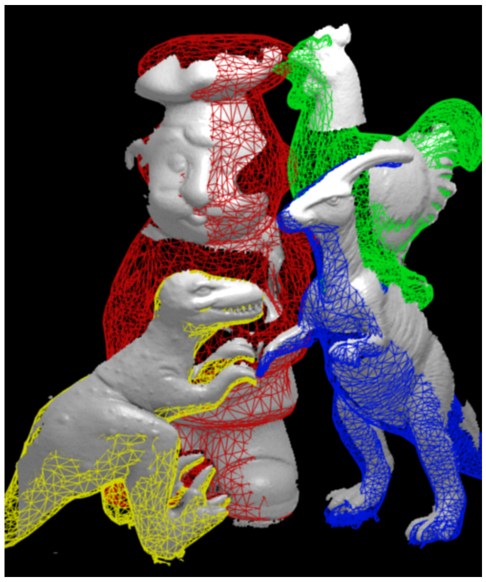
Fig. 5.22 shows the corresponding results, aggregated over 10.000 scenes. Overall, occlusions up to 50% and rotations up to $\approx 80^\circ$ were handled perfectly.

### 5.4.2 Real Data

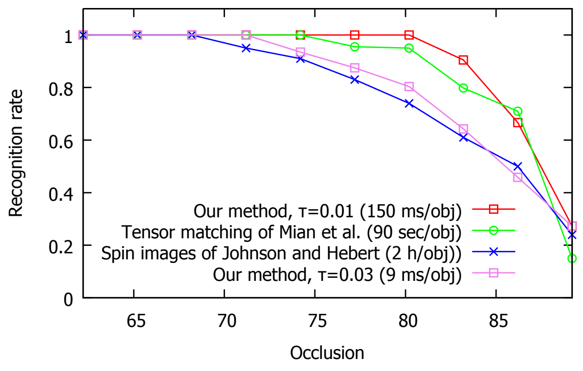
We now present the results on real data. We first present quantitative evaluations and comparisons with previous works and then show qualitative results on the scenes we acquired using a laser scanner.

**Quantitative evaluation** Our method was evaluated against the dataset provided by Mian *et al.* [92, 93], which consists of 50 scenes taken with a Minolta range scanner. Each scene contains four or five of the objects shown in Fig. 5.18 with known ground truth. In the original comparison, the rhino was excluded, because

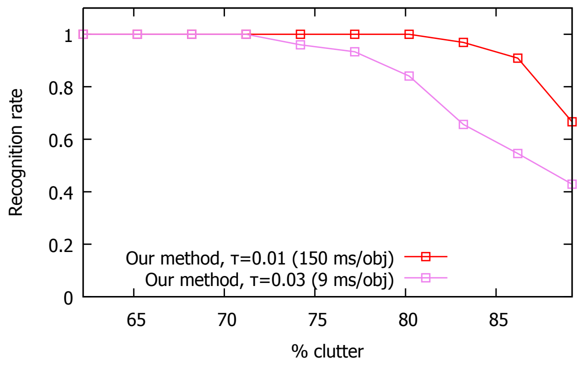
(a)



(b)

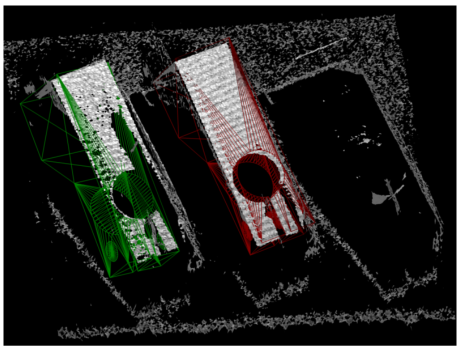


(c)

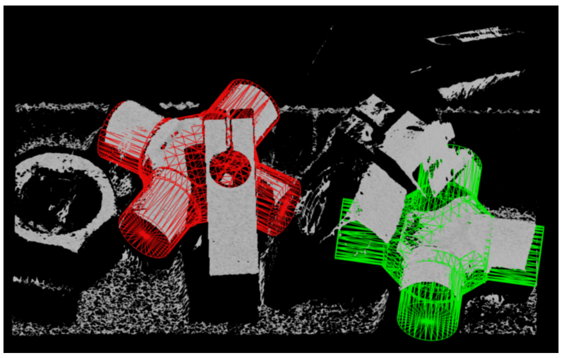


(d)

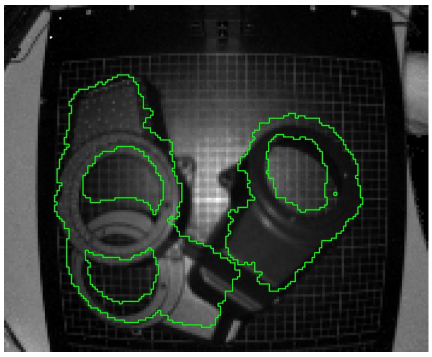
Figure 5.23: (a) Example scene from the dataset of Mian *et al*. [92] (b) Recognition results with our method. All objects in the scene were found. The results were not refined. (c) Recognition rate against occlusion of our method compared to the results described in [92] for the 50 scenes. The sample rate is $\tau_d = 0.01$ with $|S|/100$ reference points for the red curve, and $\tau_d = 0.03$ with $|S|/50$ reference points for the magenta curve. (d) Recognition rate against clutter for our method.



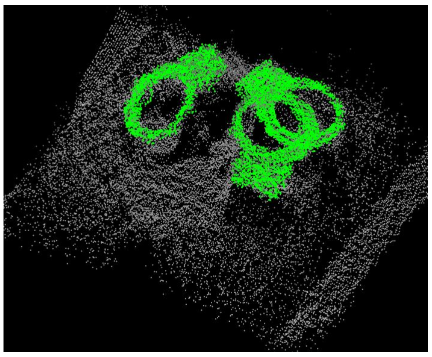
(a)



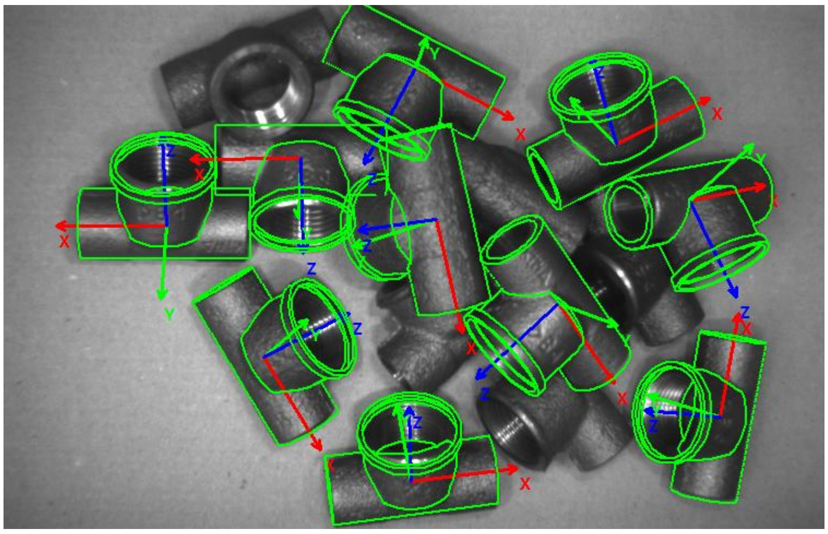
(b)



(c)



(d)



(e)

Figure 5.24: **Qualitative matching examples.** (a),(b) Noisy scenes taken with a laser line scanner. The unrefined results are shown as colored wireframes. The scenes show missing data, clutter, occlusion, and noise. (c) Refined matches in an unfiltered, highly noisy scene acquired with a commercial time-of-flight sensor. The results are projected back into the intensity image. The backprojected boundaries are jagged since the model was a template taken from a reference scan. (d) shows the scene and results in 3D; note the high amount of noise. The total runtime for matching and refinement was 37 ms. (e) Multiple detected and refined pipe joints in a scene acquired with a four-camera stereo system. The total runtime, including refinement, was 151 ms.

the spin images failed to detect it in any scene. We did the same to allow direct comparison with this prior work. Each object was searched in every scene using our method, and the pose with the best score from the pose clustering was then compared with the ground truth.

We did two detection runs with varied sampling rate $\tau_d$ to test its influence on the detection rate and the runtime. Fig. 5.23a and 5.23b show an example scene with results. Fig. 5.23c shows the results for our two runs compared with the data of the spin images and the tensor matching from [92]. The parameters of spin images were set to yield maximum performance, resulting in a very large runtime.

We found that the dataset is particularly challenging for our method due to high-frequency wrinkles on some of the object's surfaces. Computing the normals before the subsampling leads to aliasing effects and results in unstable normal directions. It is therefore important to recompute the normals after the sampling step.

For $\tau = 0.01$, the recognition rate is 97.0% of all objects with less than 84% occlusion, outperforming both tensor matching of Mian *et al.* (96.6%) and spin images (87.8%) while being significantly faster. The values relative to the 84%-boundary was taken from [92] and is given for comparability. Note that similar to the compared work, we did not post-process the poses with ICP, but used only the pose with the most number of votes.

Another advantage of our method is the possible trade-off between speed and recognition rate: For $\tau_d = 0.03$, our recognition rate drops to 89% for objects with less than 84% occlusion. However, the matching was more than 15 times faster and took less than 9 ms per object instance. The recognition rate still exceeds that of spin images. The recognition rate w.r.t. clutter is similar. Note that for one object we try to recognize in the scene, all the other objects are considered as clutter.

**Qualitative results** To show the performance of our algorithm concerning real data, we took a number of scenes with different sensors (a self-built laser scanning setup, a time-of-flight camera, and a multi-camera stereo setup). Fig. 5.24 shows several scenes and results. We did not pre-process any of the acquired point clouds (such as outlier removal or smoothing). The objects were matched despite significant clutter, occlusion and noise in the scenes. The resulting poses seem accurate enough for object manipulation, such as pick and place applications.

### 5.4.3 Feature Distribution

An important aspect for the performance of the local voting scheme is the distribution of the point pair features, *i.e.*, the distribution of $|H(\mathbf{f})|$. If a point pair feature $\mathbf{f}$ is detected in the scene, $|H(\mathbf{f})|$ votes need to be cast by the voting scheme. Objects with many long lists can thus be slower to match than objects with only short lists.

An extreme example is an object that consists of only a single planar patch. Since all points are on the same plane and have the same normal direction, the angles of all point pair features will be identical and the features will differ only by the distance of the two points:

$$\mathbf{F}(\mathbf{p}_1, \mathbf{p}_2) = (|\mathbf{p}_1 - \mathbf{p}_2|, \pi/2, \pi/2, 0) \tag{5.36}$$

As a result, all point pairs with the same distance have the same feature. Similar, spheres and cylinders contain many point pairs with similar configurations.

To illustrate how this affects performance, Fig. 5.4a shows the distribution of point pair features for several different objects. Note how objects that contain large planar sides, spherical or cylindrical areas have a worse distribution than free-form objects with a more distinctive geometry. Fig. 5.4b shows the matching time when detecting an object in a scene that contains only itself: Matching objects that contain symmetries is slower than matching objects without symmetries, given scenes of the same relative size.

Note that the performance costs also depend on how often point pairs with long lists appear in the scene. Besides the object itself, background clutter might contain such pairs. In real-world applications, the most common case are planes in the background (walls, floor, table surfaces) when matching objects with large planar sides. Intuitively, the object might be detected anywhere in one of the planes, by attaching its planar side to the plane. As a practical workaround, it might make sense to detect and remove large planar patches before matching the object.

Sec. 7 shows how the symmetries of primitive objects such as planes, spheres and cylinders can be used by a modified voting scheme that avoids above's performance regressions.

### 5.4.4 Timings and Counts

To asses the runtime impact of the different steps of the detection pipeline (Fig. 5.17), we measured the exact runtimes of all steps in the pipeline for an exemplary scene. The measurement was done in the scene shown in Fig. 3.6 (p. 41). Additionally, the number of reference points and pose candidates for each step is given.

Fig. 5.25 shows the corresponding timings. The two refinement steps clearly dominate the runtime, followed by the voting step. However, all three benefit almost linear from parallelization. A notable exception is the full ICP, which benefits slightly less than optimal. This is because it is limited by memory and cache throughput – the random access pattern of the voxel-based nearest neighbor structure is quite memory intensive – and because the remaining matches at that

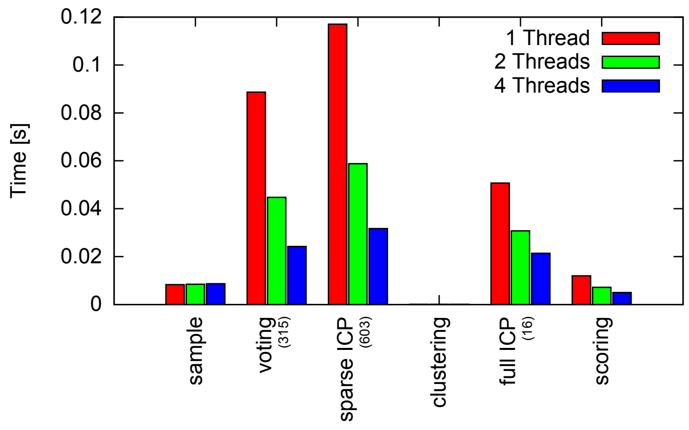


Figure 5.25: Runtimes of the different steps of the detection pipeline. **Times:** Most of the time is spent for voting and sparse ICP. Since the full ICP is performed only for a few results, its impact on the total detection time is reasonable. Note that the sampling, the clustering and the scoring step have little impact. **Threads:** Most steps, especially the time-critical ones, benefit from parallelization. Note that the full ICP benefits slightly less than optimal. **Counts:** 315 reference points were processed for voting, resulting in 603 pose candidates that were sparsely refined. After pose clustering, 16 candidates remained that were accurately refined and scored.

step are distributed less than optimal to the cores. Note also that the sampling step is not parallelized, as discussed in Sec. 2.3.

To get an intuition of the numbers involved, Fig. 5.25 also shows the number of reference points (315), the number of pose candidates after voting (603) and the number of pose candidates after clustering (16). Note that even though there were more than 35 times as many poses to be refined in the sparse ICP, it still took only about twice as long as the full ICP. Finally, after removing poses with a score less than 0.3, a total of 9 results was returned.

## 5.5 Conclusion

This chapter introduced a novel approach for detecting rigid 3D objects in 3D point clouds. It builds upon a voting scheme, similar to the Hough transform, that operates on a local, data-driven, restricted parameter space that reduces the dimension of the accumulator space to three. The voting uses 3D point pairs as features, which are fast and robust to compute and to match. The voting parameters are computed in a decomposed manner, allowing for an efficient inner voting loop. The results of the voting scheme are additionally refined and clustered in a detection pipeline. The method is fast, generic, robust, allows detection multiple object instances at once and returns the optimal (in terms of surface overlap) local pose. Experiments show that the method outperforms prior art on a standard dataset.

# 6 Rigid Object Detection in Multimodal Data

This chapter introduces a variant of the voting scheme and the detection pipeline introduced in Sec. 5: The detection scheme is extended to find rigid objects in multimodal RGB-D data, *i.e.*, in data where both an intensity or color image and a depth image of the scene are available. For this, the 3D point pair feature is replaced by a multimodal feature that simultaneously describes the observed 3D surface as well as edges found in the 2D image. The feature is invariant against the distance of the object from the camera, against in-plane rotation and against perspective distortion.

The experiments show that using this kind of multimodal information leads to significant improvement for objects that have planar sides, which the method of Sec. 5 would find in background clutter. Compared to prior art, the method has a comparable performance for objects that are completely visible but performs significantly better for partially occluded objects.

Parts of this chapter were previously published in [41].

## 6.1 Introduction

**Edges in 3D** The use of edges for object detection in 2D intensity images has a long tradition and has proven to be a powerful feature in object detection [106, 54, 131, 67]. By contrast, remarkably little work [135, 130, 74] has been published on using edges or depth discontinuities in range images for such tasks. This is probably due to a number of challenges that are unique to 3D edges. Most notably, range sensors tend to fail exactly at or around such depth discontinuities. Triangulating reconstruction methods, such as stereo or structured light, suffer from local occlusion around such edge points. Other methods, such as time-of-flight, tend to smooth over edges and introduce veil points, *i.e.*, points at 3D positions that do not correspond to any real 3D points in the scene. Such effects make it difficult to detect and accurately localize depth discontinuities in range images. By contrast, edges in intensity images can be detected and measured typically with sub-pixel precision, but it is in general impossible to distinguish between texture and geometric edges.

We propose a method that combines the high accuracy of the intensity edges with the geometric expressiveness of range information. For this, edges are extracted from the intensity image – yielding a highly accurate position and direction – and filtered using the range image to obtain only edges that correspond to depth discontinuities, the *geometric edges*. These geometric edges are combined with the 3D data from the range image to form a novel *multimodal point pair feature descriptor* that combines intensity and range information in a scale and rotation invariant way. The object's appearance from different viewpoints is described in terms of these features, which are stored in a model database. This model database allows efficient access to similar features and captures the overall appearance of the object. In the online phase, the multimodal features are extracted from the scene and matched against the model database. The local voting scheme of Sec. 5 is used to group these matches and find the pose that simultaneously maximizes the overlap of the object's silhouette to the detected geometric edges as well as the overlap of the model surface and the 3D surface of the scene. The resulting pose candidates are filtered using clustering and non-maximum suppression, and the final result is refined using the iterative closest point (ICP) algorithm to obtain a highly accurate pose.

Using only geometric edges instead of all edges that can be detected in the intensity image is mostly a performance issue. Since there are in general fewer geometric than texture edges, the voting needs to process less features, especially in the presence of background clutter. It also allows to train based on untextured CAD models and improves the robustness, as cluttering texture is completely ignored.

The advantages of the proposed method are numerous: It is able to localize textured and untextured objects of any shape and finds the full 3D pose of the object. Multiple instances of the object can be found. The method can be trained either using a CAD model of the object or using registered template images taken from different viewpoints. When using templates, one template per viewpoint is sufficient since the method is invariant against scale changes and in-plane rotations. The method also shows high robustness to background clutter and occlusions.

**Contribution** The main contributions of this chapter are in particular

1. A multimodal geometric edge extractor that combines the accuracy of intensity edges with the expressiveness of depth edges.
2. A viewpoint dependent multimodal point pair feature that simultaneously

describes the object's surface and its silhouette, and that is invariant against scale changes, in-plane rotation and perspective distortions.

3. The integration of the above feature into the voting scheme of Sec. 5.

The proposed approach is evaluated both quantitatively and qualitatively and compared against other state-of-the-art methods. It shows comparable results and higher robustness with respect to occlusions. In the remainder of this chapter, we will first discuss related works, describe our method, and finally present our results.

## 6.2 Related Work

A more detailed list of methods related to 3D object detection can be found in Sec. 5.2. Here, we focus on multimodal object detectors that combine depth and intensity.

Stiene *et al.* [135] proposed a detection method in range images based on silhouettes. They rely on a fast Eigen-CSS method and a supervised learning method. However, their object description is based on a global descriptor of the silhouette and is thus unstable in the case of occlusions. They also require a strong model of the environment, which does not generalize well. Steder *et al.* [130] use an edge-based keypoint detector and descriptor to detect objects in range images. They train the descriptors by capturing the object from all directions and obtain good detection results in complex scenes. Compared to their approach, the proposed method is more robust as it does not require a feature-point detector that relies on object parts with corner-like characteristics. Wu *et al.* [153] used a perspective correction based on 3D data similar to the correction presented in this chapter.

Hinterstoisser *et al.* [65] proposed a multimodal template matching approach that is able to detect textureless objects in highly cluttered scenes but is sensitive to occlusion and does not recover the 3D pose of the object. Compared to their approach, the approach presented in this chapter is more robust to occlusions and, owing to the scale- and rotation-invariant feature descriptor, requires fewer template images. Sun *et al.* [138] use multimodal information to simultaneously detect, categorize, and locate an object. However, while working well in many scenarios, the approach requires large training datasets. Lai *et al.* [80] propose a distance-based approach for object classification and detection in multimodal data and provide a large evaluation dataset. However, they also do not recover the pose of the object and show no results for clutter that is close to the target object.

A small number of edge detectors for range images were proposed, such as [130, 74]. However, designing a generic edge detector for range images is a very difficult task, mostly because different sensors tend to behave very

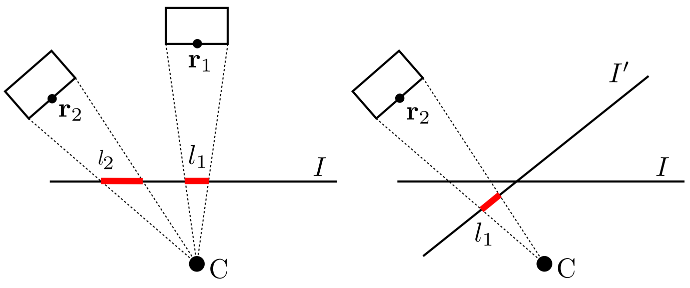


Figure 6.1: Left: Even if the object is seen from the same direction and from the same distance, lengths and angles appear distorted in the image plane. Right: We apply a perspective correction by reprojecting onto a new image plane $I'$ with the same focal distance, but orthogonal to the line of sight to the reference point $\mathbf{r}$.

different at or around edges. As discussed above, different range sensors, such as time-of-flight cameras, stereo systems, structured light, laser triangulation, depth-from-focus, or photometric stereo exhibit very different characteristics in terms of noise, missing data, occlusion and smoothing.

Choi *et al.* [29] proposed to detect and use edges and line segments in range images, and to form point pair features that include those edges or line segments. This is similar to the approach in this chapter. Their approach is, however, limited to range sensors that provide very accurate information at edges, as they need to both detect the edge direction and the depth at the edge. Additionally, their approach is limited to objects that have articulated edges.

Compared to the rigid object detector from Sec. 5, which uses 3D information only, the multimodal approach is more robust when detecting objects that appear similar to background clutter. This is often the case if the object's surface contains large planar patches: Since the method from Sec. 5 does not take the object boundaries or the viewpoint-dependent appearance into account, such objects are often detected in walls or tables.

## 6.3 Method

The objective is to detect a given 3D object and determine its pose in a multimodal RGB-D image. Here, RGB-D stands for sensors that provide both, an intensity image (RGB) and a depth image (D) from the same viewpoint.[1] The proposed method does not use color, but only edges. It can thus work also with multimodal sensors that provide a grayscale instead of a color image. However, since multimodal sensors are often denoted as RGB-D sensors, we will stick to this notation.

[1]Note that sensors often have slightly different viewpoints for the intensity and depth sensor. However, if their relative pose is known, the 3D data from the depth data can be re-projected into the intensity image to simulate a common viewpoint.

We assume that the 3D model of the object we want to detect and localize is available to us, either as a 3D CAD model or reconstructed from multiple, registered RGB-D images. The scene in which we search for the object of interest is captured with an RGB-D sensor and may contain clutter and occlusions.

The proposed approach differs from the voting scheme and the detection pipeline introduced in Sec. 5 by the choice of the feature. Instead of using the point pair feature from Sec. 5.3.1, a multimodal feature is used that combines 3D surface information and 2D contour information. The remaining parts of the detection pipeline – notably the voting scheme over a local parameter space, the pose clustering and the optional refinement – are left untouched. Note that one could use a refinement method that optimizes both modalities simultaneously instead of ICP, which optimizes only the 3D modality. However, such a method is outside the scope of this work.

In the reminder of this section we will first introduce our novel multimodal feature and then discuss its use for object model description and then detection and localization.

### 6.3.1 Multimodal Feature

Let $M$ be the oriented 3D point cloud of the model we want to detect and localize. The scene in which we search the object of interest is an RGB-D image composed of an intensity or color image $I_C$ defined for domain $\Omega_C$ and a range or depth image $I_R$ defined for domain $\Omega_R$. As described further below, we extract a set of geometric edge points $\Omega_E \subset \Omega_C$ from both modalities. In practice, the range image must be calibrated so that we know the metric measurement of the scene. Our principal intuition for the creation of our multimodal feature is that the intensity image provides accurate information at depth discontinuities, such as the object's contour, where depth sensors tend to fail, while the depth sensor provides information on the inner surface of the object where, in the absence of texture, the intensity image provides little or unreliable information. Therefore, intensity image and range image complement each other and our feature combines the stable information from both modalities. The feature pairs a reference point $\mathbf{r} \in \Omega_R$ from the range image, selected from the visible part of the object, and a point $\mathbf{e} \in \Omega_E$ from the set of geometric edge points. As the geometric boundaries, and thus the geometric edges, of an object depend on the viewpoint, our feature is inherently viewpoint-dependent.

**Perspective Correction** The appearance of an object in a projective image depends on the direction from which it is seen, on the distance to the projection center, and on the position of the projection in the image plane. The perspective distortion due to the position in the image plane disturbs measurements of distances and angles. To be more robust against such distortions, we employ a perspective

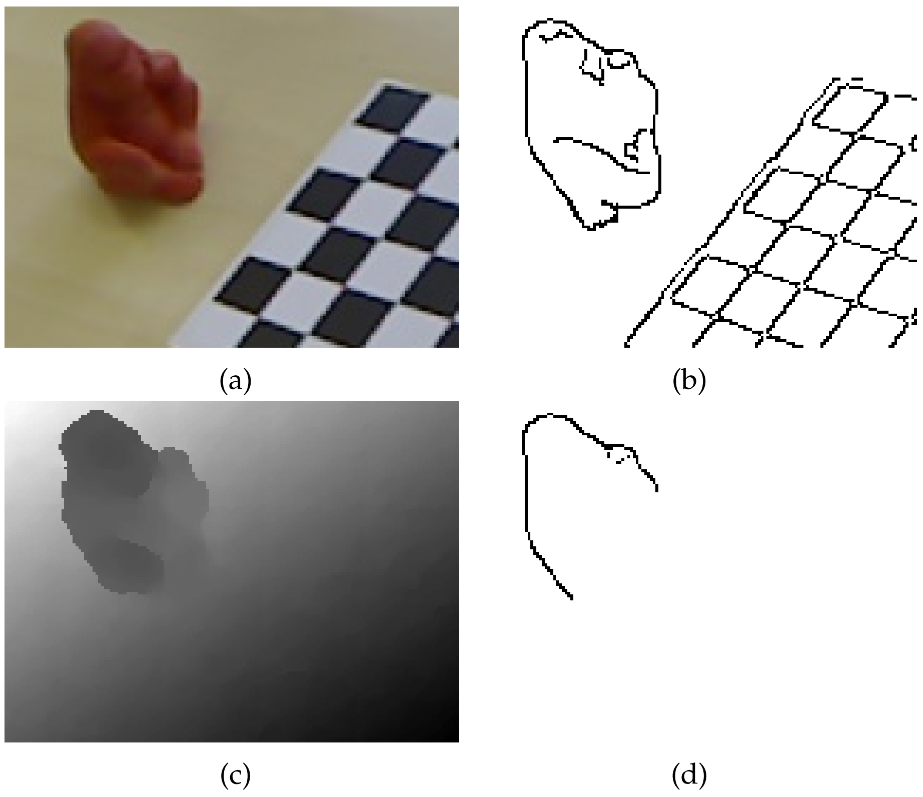


Figure 6.2: Example of the geometric edge detector. (a) Original color image. (b) Edges extracted from the color image. (c) Range image. (d) Filtered geometric edges: Color edges without depth discontinuity perpendicular to the edge direction are removed. Only geometric edges remain, which are localized with high accuracy in the color image.

correction step that reprojects the edge $\mathbf{e} \in \Omega_E$, the reference point $\mathbf{r} \in \Omega_R$, and its normal $\mathbf{n}_r$ onto a new image plane. Without loss of generality, we assume the camera center to be in the origin, *i.e.*, the viewing direction towards $\mathbf{r}$ is $\mathbf{v}_r = \mathbf{r}/|\mathbf{r}|$. The new image plane onto which we project is defined as the plane perpendicular to $\mathbf{v}_r$ and with some fixed focal distance $f$ from the projection center.

The reference point $\mathbf{r}$ is thus projected into the center of the new image plane, and the visible features appear as if seen in the center of the image. Fig. 6.1 depicts this correction step. For clarity, we continue to write $\mathbf{e}$ and $\mathbf{r}$ even if the corrected values are meant. This undistortion is performed both in the online and the offline phase and boils down to a homography that is efficiently applied on-demand to each edge point.

**Geometric Edge Detection** Visible edges in an intensity image can be categorized into texture edges, which are found mainly on the inner parts of a surface, and geometric edges that appear due to depth discontinuities in the scene. The latter occur mainly on the occluding boundaries of objects, but can also appear on inner parts of the object. To localize an object, the proposed method uses only the object's geometric edges for several reasons. First, every object, textured or untextured, has a silhouette and thus a geometric boundary. The geometric edges are therefore a very generic feature. Second, there are typically fewer geometric edges than texture edges in cluttered scenes, such that fewer features need to be processed. Third, geometric edges complement the surface information from the depth sensor as described above. And finally, geometric edges are easy to detect in RGB-D images, as described below.

To obtain geometric edges, we first detect edge pixels **e** with gradient direction $\mathbf{e}_d$ in the intensity image $I_C$ using the Canny color edge detector [23]. The detected edges are then filtered using the depth image to obtain the geometric edges. The filter computes the minimum and maximum depth value on a line segment along $\mathbf{e}_d$,

$$D(\mathbf{e}) = \max_{\alpha \in [-s,s]} I_R(\mathbf{e} + \alpha \mathbf{e}_d) - \min_{\alpha \in [-s,s]} I_R(\mathbf{e} + \alpha \mathbf{e}_d) \tag{6.1}$$

The edge point is classified as geometric edge if $D(\mathbf{e})$ exceeds a certain threshold. The threshold should be larger than the expected depth noise of the sensor and is in practice set to 1-3 cm for images captured with a Kinect-like device. In our experiments, the scan range $s$ was set to 3 pixels.

This proposed filtering of geometric edges is computationally very efficient, since it needs to evaluate only a few pixels per detected color edge point. It is also robust w.r.t. veil points and noise. The orientation of an intensity edge depends on the local color gradient. We re-orient the direction of the edge such that the gradient $\mathbf{e}_d$ points out of the object, *i.e.*, from the surface closer to the camera towards the surface further away. Fig. 6.2 shows an example of the geometric edge detection.

**Multimodal Feature** The multimodal point pair feature describes the geometric relation between an edge point **e** and a depth point **r** in the perspectively corrected image. It is described by $F(\mathbf{e}, \mathbf{r})$ using a four-dimensional feature vector as depicted in Fig. 6.3. The feature vector is defined as $F(\mathbf{e}, \mathbf{r}) = (d(\mathbf{e}, \mathbf{r}), \alpha_d, \alpha_n, \alpha_v)$ and contains

- the metric distance $d(\mathbf{e}, \mathbf{r}) = Z(\mathbf{r})|\mathbf{e} - \mathbf{r}|/f$ of the two points. $f$ is the focal length of the projection system and $Z(\mathbf{r})$ is the depth of the reference point **r**. The scaling factor $Z(\mathbf{r})/f$ transforms the measurement from pixels to metric units, making it invariant against the distance of the point pair from the camera and against the focal length, *i.e.*, scale invariant. Note that this allows using sensors with different focal lengths for training and matching;
- the angle $\alpha_d = \measuredangle(\mathbf{e}_d, \mathbf{e} - \mathbf{r})$ between the difference vector of the two points and the edge gradient $\mathbf{e}_d$;

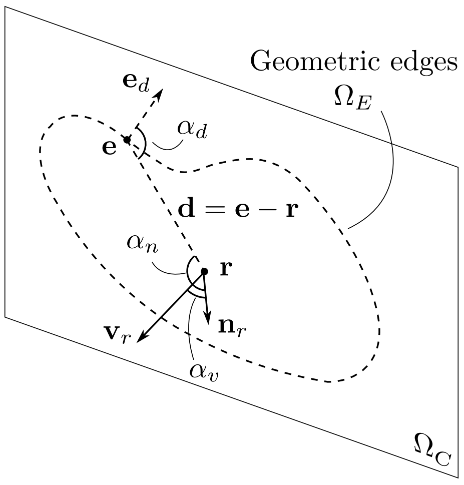


Figure 6.3: Description of the used feature descriptor. Dashed vectors and lines live in the image plane, while solid vectors are in 3D. Our feature uses the angles $\alpha_r$, $\alpha_n$, and $\alpha_v$ as well as the scaled distance $Z(\mathbf{r})|\mathbf{r} - \mathbf{e}|/f$. All four components are invariant against in-plane rotations and scaling.

- the angle $\alpha_n = \measuredangle(\mathbf{n}_r, \mathbf{e} - \mathbf{r})$ between the difference vector and the normal vector; and

- the angle $\alpha_v = \measuredangle(\mathbf{n}_r, \mathbf{v}_r)$ between the normal vector and the direction towards the camera.

The range of $\alpha_d$ is $[0; 2\pi]$, while that of $\alpha_n$ and $\alpha_v$ is $[0; \pi]$.

This feature vector is designed to depend only on the viewpoint of the camera w.r.t. the object. It is most notably invariant against scale changes, *i.e.*, against the distance of the object from the camera, against rotations of the object around the viewing direction (in-plane rotations), and against perspective distortions. This design allows to train the proposed method using only a single template per viewing direction. Multiple scales and rotations, such as in [65], are not required.

### 6.3.2 Model Description

In the offline phase, a model description that contains the appearance of the object from various viewpoints is built. This is done by rendering the model from viewing directions sampled on the sphere around the object or by using templates acquired by the user. The model description is represented by a hash table that maps quantized multimodal feature vectors to lists of multimodal point pairs with similar feature vectors. This is similar to the descriptor used in Sec. 5.3.2, but using the proposed, viewpoint dependent multimodal feature. The hash table allows constant-time access to similar features on the object.

The following steps are performed to create the model description:

1. Create a set of model reference points $R \subset M$ by uniformly sampling the model (see Sec. 2.3).

2. Select a set of viewing directions that contain all directions from which the object can be seen in the online phase. In practice, we uniformly sample $\approx 300$ viewpoints from the unit sphere. Using less lead to a decreased performance, while using more than 300 viewpoints lead to no measurable improvement of matching performance. This is due to the remove of duplicate features in the voting lists (see Sec. 5.3.5): viewpoints that are very similar to an already used one lead to identical features, which are then removed.

3. Obtain the object's appearances $I_C$, $I_R$ from the different viewing directions. This is done either by rendering the object from the selected directions, or by using template images acquired by moving an RGB-D sensor around the object and registering the views.

4. For each template, detect the geometric edges $\Omega_E$ on the object as described above.

5. For each geometric edge point $\mathbf{e} \in \Omega_E$ and each model reference point $\mathbf{r} \in R$ visible in the corresponding template, compute the multimodal feature $F(\mathbf{e}, \mathbf{r})$, quantize it, and store it in the hash table that describes the model.

The sampling parameters of the angle and distance coefficients of the feature depend on the expected noise level. In practice, they are set to 12° for the angles and 3% of the object's diameter for the distance value.

### 6.3.3 Voting Scheme

The online phase uses the voting scheme presented in Sec. 5.3.4 with two modifications. First, instead of using the 3D point pair feature, the multimodal point pair feature is used. The reference point corresponds to the 3D point of the multimodal point pair, while the edge points are extracted the same way as in the offline phase. Second, the closer the object is to the camera, the larger it appears and the more edges are visible, leading to a higher score in the voting scheme. This bias is removed by multiplying the number of votes with the distance of the scene reference point from the camera.

The pose candidates that are created by the voting scheme are processed in the same pipeline as described in Sec. 5.3.9, using non-maximum suppression and optionally ICP as pose refinement.

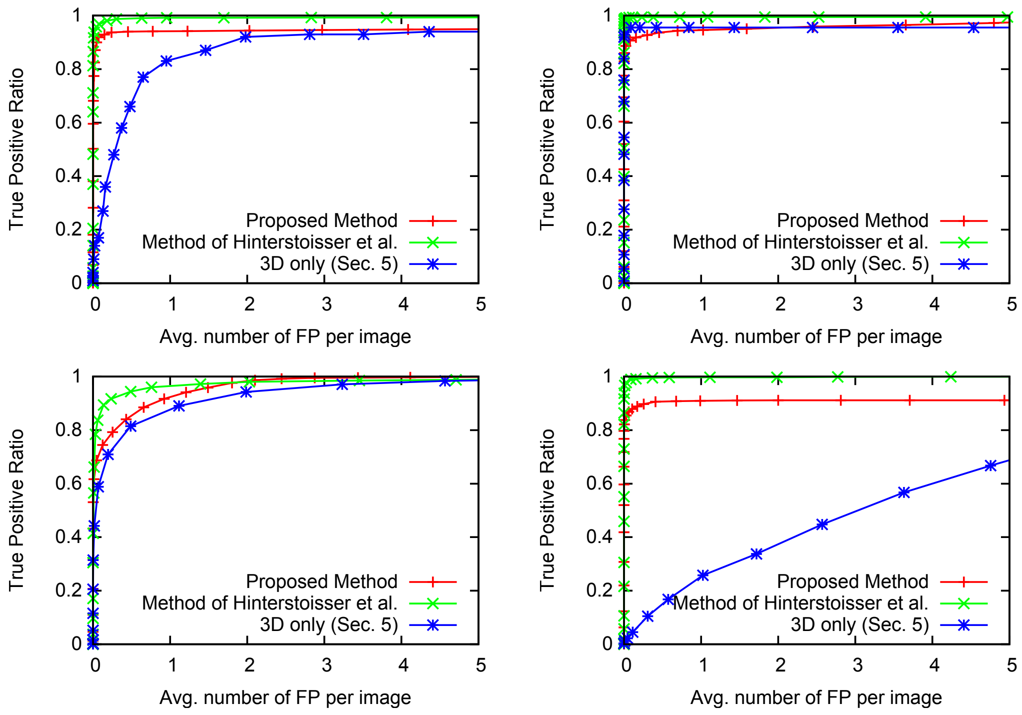


Figure 6.4: From top left to bottom right: Detection results for the APE, DUCK, CUP, and CAR dataset from [65] for the proposed and the compared methods. Note that the method of Hinterstoisser *et al.* does not recover the object's pose, as opposed to the other two methods. The duck has a rather unique surface and is detected by all three methods with a high detection rate. The car contains planar patches similar to background clutter, leading to misdetections of the method of Sec. 5 that uses the 3D information only, while the two multimodal approaches keep a high recognition rate.

## 6.4 Experiments

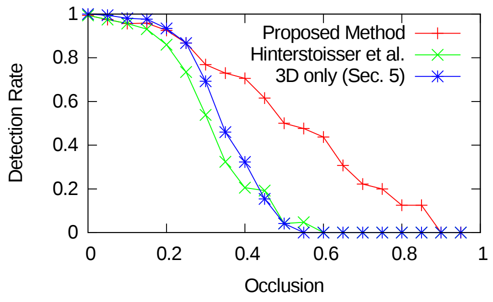
We evaluated the proposed method quantitatively and qualitatively on multiple datasets and compared it against state of the art. All datasets were captured using a Microsoft Kinect or a Primesense sensor to obtain an RGB and a depth image with resolution 640 $\times$ 480. Both modalities were calibrated and registered.

All models were available as a CAD model. The scene reference points were selected by uniformly sampling the scene with a distance of 4% of the object's diameter. All tests were run on an up-to-date computer using an unoptimized C implementation. The evaluation took 2-10 seconds per scene, mostly depending on the scene size and the number of detected geometric edges. We believe that an improved implementation would speed up the method by an order of magnitude.

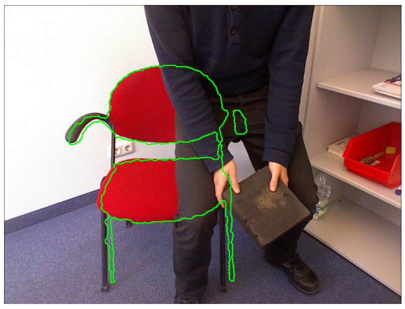
### 6.4.1 Quantitative Evaluation

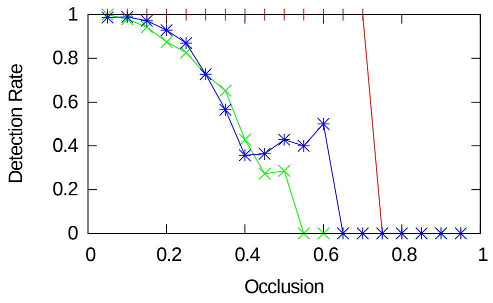
**Dataset Hinterstoisser *et al.*** We first evaluated the method on several sequences from Hinterstoisser *et al.* [65], namely the APE, DUCK, CUP and CAR sequence.

(a)



(b)


(c)



(d)

Figure 6.5: (a) Example image of the artificial occlusion that covers the ape. (b) Detection rate vs. occlusion for the occluded ape sequence. The occlusion measures how much of the original ape surface visible in the image is occluded. (c) Example image showing the occlusion on the real-world occlusion dataset and the detection result of the proposed method in green. (d) Detection rate vs. occlusion for the occluded chair sequence.

The CAMERA and the HOLEPUNCHER sequence were not evaluated because no CAD model was available. Each sequence contains 255 template images that were used for learning the features and over 2000 evaluation images. Note that the template matching used in [65] does not recover the pose, as opposed to our method. Nevertheless, to allow a comparison between both methods, we employ the criterion of [65] to classify the correctness of matches. This criterion compares the bounding box of the ground truth and the match. Note that this classifies matches at the correct position but with an incorrect rotation as correct. However, we found that such incorrect classifications were rare for our approach. Fig. 6.6 shows example scenes and detections. Fig. 6.4 shows the detection rates for our proposed method, the method of Hinterstoisser *et al.* and the method of Sec. 5.

Overall, our method performs slightly worse than the method of Hinterstoisser *et al.*, but still reaches quite high recognition rates. A manual inspection of the scenes where our method failed shows that most missing detections were due to a failure of the 2D Canny edge extractor. This extractor failed in scenes with high motion blur and in areas with little contrast, *i.e.*, when object and background color were similar. Compared to the localization method of Sec. 5, the multimodal method performs approximately equal for the duck and cup

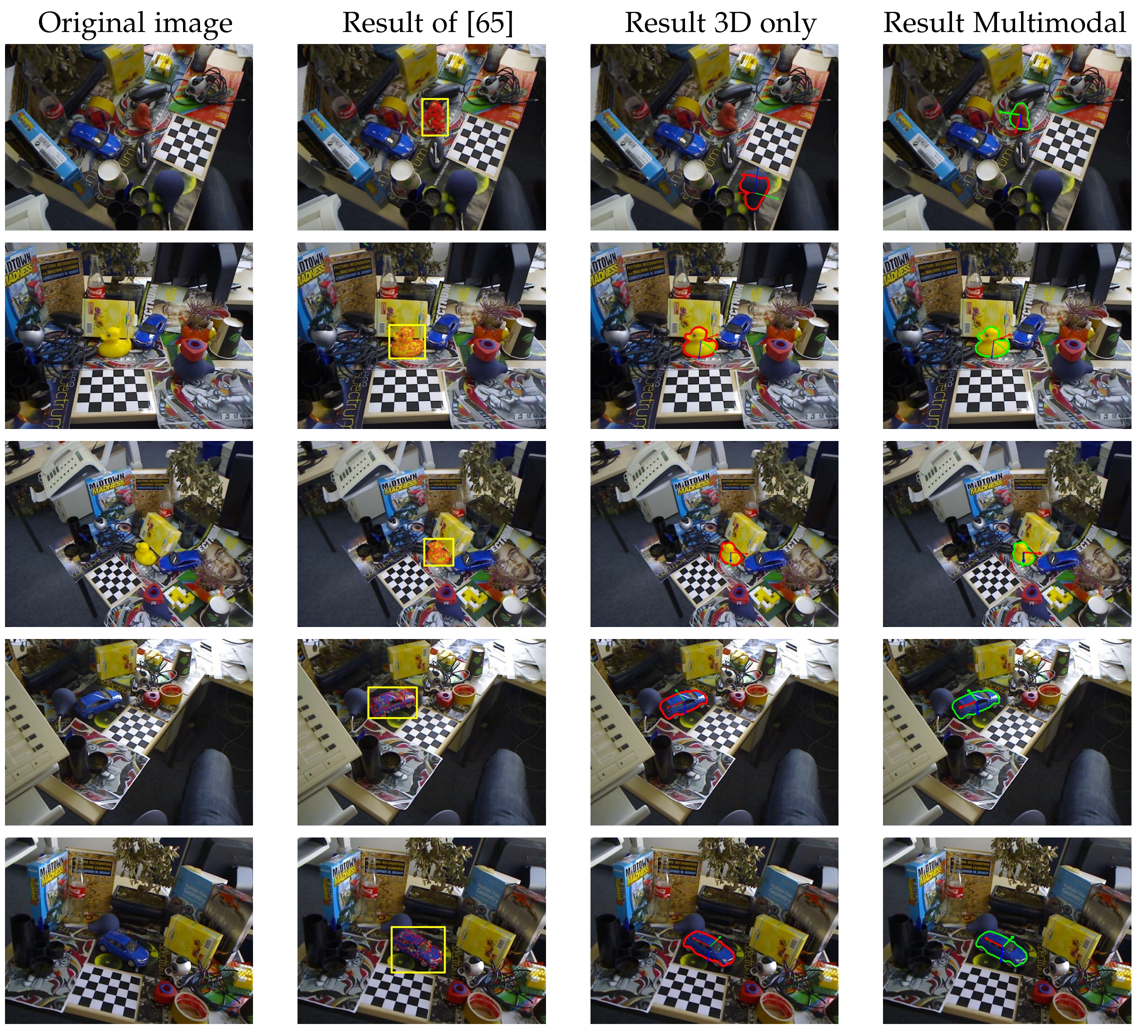


Figure 6.6: Detection results for several scenes from the dataset of [65]. From left to right: Original scene, detection result of [65], detection result of the 3D-only method of Sec. 5 (in red), multimodal result (in green). Note that the method of Hinterstoisser *et al*. does not recover the object's pose, but only the bounding box of the best matching template.

sequence but significantly better for the ape and car sequence. This is mostly due to the rather flat back of the ape and the large planar parts of the car: The method of Sec. 5 optimizes the surface overlap of model and scene, leading to false positives if large parts of the object are similar to clutter. The method proposed in this chapter, which optimizes both surface and silhouette overlap, is able to correctly remove such false positives.

Another difference is that the approach in [65] uses all edges in the templates, including texture edges. This might improve the robust of their approach compared to ours, where only contour edges are used. Note, however, that the dataset is comprised of objects that have little to no texture.

**Occlusion** We additionally evaluated the three methods against partial occlusion of the target object using two datasets. For the first dataset, one of the images from the ape sequence was disturbed by artificially occluding different parts of the ape in both the RGB and the depth image. A total of 256 images with varying amounts of occlusion was created this way. The detection rates of all three methods and an example image are show in Fig. 6.5 (a), (b). Note that we set the parameters of the method of Sec. 5 such that similar timings as with the new method were obtained. As explained in Sec. 5, higher detection rates can be achieved by allowing the method to run longer.

For the second dataset, we used varying amounts of real occlusion of a chair that was put in a fixed position with respect to the sensor. For the model creation, a reference image without occlusion was used. Fig. 6.5 (c), (d) show an example image and the resulting detection rate for this dataset.

The proposed method clearly outperforms both the 3D only matching and the method of [65] in case of non-trivial occlusion. For the method of Hinterstoisser *et al.*, occlusion of certain key regions of the object that contribute the most to the detection lead to drastically reduced detection rates. On the contrary, our method treats all edge and inner regions equally. The method of Sec. 5 suffers from a similar effect, where for larger occlusions the remaining surface parts of the object look more similar to background. Since the proposed method optimizes both surface and geometric edge overlap, significantly larger occlusions are necessary to reduce the detection rate.

### 6.4.2 Qualitative Evaluation

The proposed method was also evaluated qualitatively with objects of different size and shape. Fig. 6.7 show several test images and detections. We found that the method performs very well even in case of occlusion and large amounts of clutter. The method works equally well and with the same speed for all tested objects. This is a major advantage over the method of Sec. 5, which deteriorates both in terms of speed and detection performance for planar objects.

## 6.5 Conclusion

This chapter introduced a novel method that detects rigid objects in multimodal RGB-D data. The method is based on the voting scheme of Sec. 5 but replaces the 3D point pair feature with a multimodal point pair feature that combines the most valuable information – accuracy of intensity edges and geometric expressiveness of depth edges - from both modalities.

The experiments show that the method is fast, robust and generic. Compared to the baseline method of Sec. 5, the modifications significantly improve performance for objects with large planar sides. Compared to prior art, the method

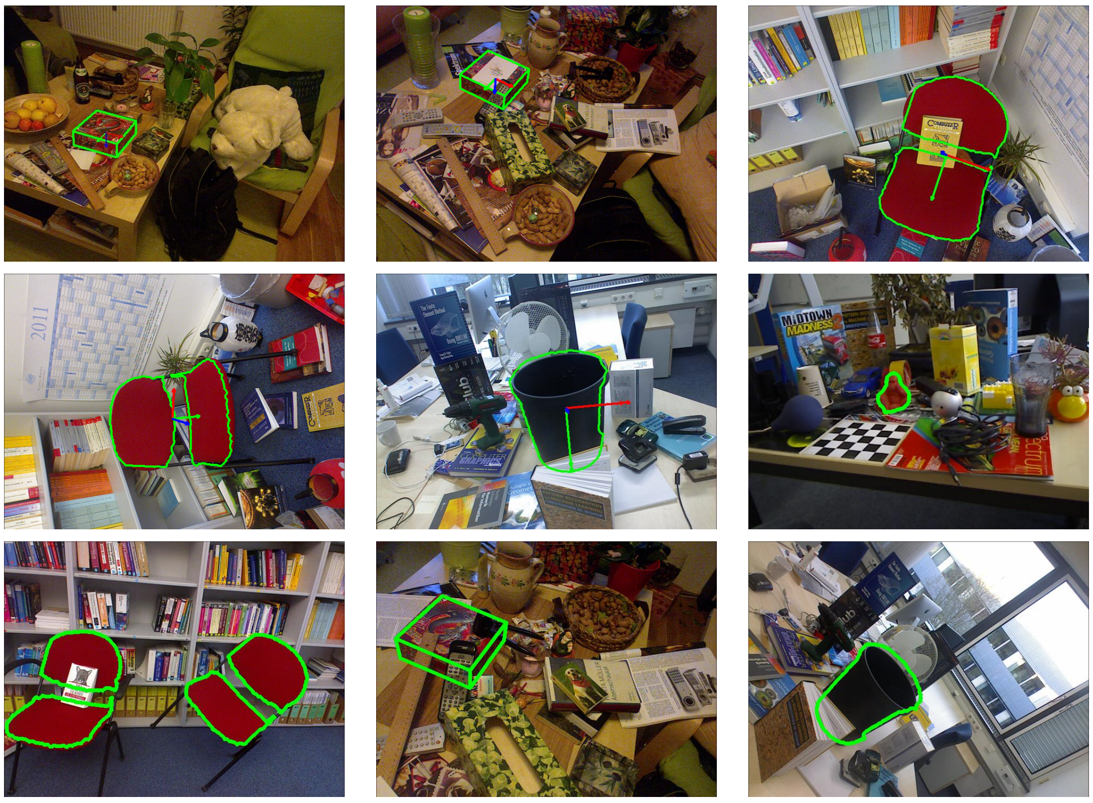


Figure 6.7: Detection results for several example scenes, showing large amounts of clutter and occlusion, multiple instances, texture-less objects and planar objects, all in arbitrary poses. The detections are outlined in green. Note that objects that were reconstructed from reference images, such as the chair, have a rather rough outline compared to objects modeled from CAD data, such as the box.

has comparable performance for well-visible objects and a significantly better detection rate for partially occluded objects.

# 7 Primitive Shape Detection in 3D Point Clouds

The segmentation and fitting of geometric primitives is often an important part of scene understanding, robotics, reverse engineering and other applications. It can help to build a high-level description of the scene by decomposing a potentially large set of 3D points into a small set of primitives. For example, a robot operating in indoor environments can segment planes to find floors, ceilings and walls. Some object detection schemes decompose the target object into primitive shapes and attempt to detect those shapes in their particular configuration in the scene ([129]). Segmentation of primitives also has applications in reverse engineering. One particular application is the detection of buildings in 3D LIDAR scans taken from planes [149, 141].

This chapter proposes a novel approach for the detection and segmentation of certain geometric 3D primitives – namely planes, spheres and cylinders – from 3D point clouds. It is based upon the local voting scheme introduced in Sec. 5, which is able to detect 3D objects of arbitrary shape. Here, we use the symmetries of primitives for three fundamental optimizations, which make the detection faster, more robust and more generic. First, the feature database can be made implicit. Second, the voting space, *i.e.*, the parametrization of the object's pose, can be optimized by removing redundancies that arise due to object symmetries. Finally, the parameter space can be extended to hold simple shape parameters, such as the radius of a cylinder or a sphere. This chapter also introduces an extension of the ICP method from Sec. 4 that refines the candidate primitives found by the voting scheme.

Parts of this chapter were previously published in [44].

## 7.1 Introduction and Related Work

Several approaches for the detection of geometric primitives in 3D point clouds were suggested in the literature. Most approaches can roughly be classified as region growing, RANSAC [52] or Hough transform-based voting schemes. Our approach is based on the latter.

While the Hough transform has promising properties, such as deterministic runtime and robustness against local disturbances that can interfere with segmentation based approaches, it has several drawbacks when applying it to primitives in 3D. First, the size of the Hough space grows exponentially with the number of parameters. For example, cylinders have four parameters for their position and one for the radius, leading to a five-dimensional parameter space. High-dimensional voting spaces, however, require significantly more memory, are slower to process as usually more votes are cast, and have an increased sensitivity due to binning. Second, it is difficult to uniformly sample the directional components of primitives such as planes (plane normal) and cylinders (main axis direction), as there is no proper uniform segmentation of the unit sphere in 3D for arbitrary sampling sizes. Attempts to work around those problems, such as removing some dimensions by projecting them, typically lead to problems with large scenes where a lot of clutter is present or that contain multiple object instances.

Our approach circumvents all the problems of the traditional Hough transform. By using the local parameters of Sec. 5.3.3, the voting space is smaller and more robust. Additionally, the directional component is implicit in the scene normals and requires no discretization. Finally, since our approach is local, it is highly robust against even with large amounts of clutter and deals well with multiple shape instances. We give a theoretical comparison between our proposed Hough-based voting scheme and RANSAC in Sec. 7.3.

**Contributions** The contributions of this chapter are in particular:

- A new Hough transform like voting scheme for the detection of geometric primitives that circumvents the problems of ‘classical’ Hough schemes for such objects, especially regarding the parametrization of directions and the size of the accumulator space;
- robust ICP-like refinement algorithms for the different primitives; and
- a corresponding detection pipeline that refines the results of the Hough transform, removes duplicates and computes a robust score for the primitives.

**Region Growing** Methods based on region growing usually start with an (over)segmentation of the scene – sometimes even up to individual points – and combine spatially neighboring segments if they describe a consistent primitive. Different kinds of segmentation, neighborhood metrics of the segments and models to be fit into the segments exist. Mörwald *et al.* [96] fit B-Splines of first or second order into adjacent segments, fusing the segments if the fit of the fused segments is better than for the two separate segments. By using B-Splines, their approach is able to segment several types of geometric primitives. Kim and Ahn [78] proposed a region growing algorithm that uses curvature analysis of local patches to automatically select the primitive type and its parameters. Nurunnabi *et al.* [104] use a robust variant of the PCA, the *Minimum Covariance Determinant*, to reduce the influence of outliers. Attene *et al.* [5] use a hierarchical region growing approach to segment a triangulated mesh into primitives, joining neighboring triangles or groups if the joint group is better approximated by a primitive than the two disjoint groups.

Region growing methods usually have favorable runtimes as the merging is often linear in the number of segments. However, especially compared to the proposed approach, they also have several drawbacks: Primitives that are made up of disconnected components (such as a plane that is separated by an occluding object) are not joined; greedy decisions during the merging of segments can lead to incorrect, yet unrecoverable decisions; several parameters need to be fine-tuned for both the segmentation and the merging steps; if the input data and the transition between two primitives is very smooth, the segmentation might fail to segment; and finally, if the input data is very noisy, the segmentation step might oversegment and the merging step might fail to merge those neighboring but noisy segments. All of these issues are resolved by the proposed method.

**RANSAC** RANSAC-based approaches randomly sample a set of points from the scene and compute a primitive that contains all points. The hypothesis is evaluated by counting how many scene other scene points it contains. Multiple such hypotheses are randomly created and the one containing the most scene points is returned. Tarsha-Kurdi *et al.* [141] compare several approaches and designed a RANSAC-based approach that integrates domain-specific knowledge of the shape sizes. They conclude that RANSAC is inferior to the Hough transform.

Compared to the proposed method, RANSAC-based methods have the disadvantage of being nondeterministic in runtime and output, a property that is often undesirable in industrial setups. Additionally, in the presence of large amounts of clutter, noise or many instances, they require careful tuning of the parameters – such as the neighborhood from which to select the random samples – in order to remain efficient.

**Voting** Finally, methods based on the Hough transform recover the parameters of primitives using a voting scheme. The parameter space is discretized into bins, and the detected features vote for parameters that explain them. Several authors proposed the Hough transform for detecting planes in bird-eye LIDAR-Scans of urban environments [149, 107]. Rabbani and Van den Heuvel [110] proposed a Hough-based approach for cylinders. Bormann *et al.* [21] gives a comprehensive review of Hough-based approaches for detecting planes in 3D point clouds. They describe the difficulty of finding a good Hough space, which is a compromise between accuracy, runtime, storage space, and robustness. In the particular case of planes and cylinders, one needs a uniform subdivision of the space of normal directions. Since that does not exist in the general case, several approximations were proposed. Bormann *et al.* propose a new one, based on an adapted sampling of the polar coordinates. While an improvement, their discretization is still nonuniform, making computation of the parameters from the features expensive. The approach proposed in this chapter completely circumvents the issue of discretizing normal directions using a data-driven, local parametrization.

**Others** Ahn *et al.* [1] proposed a semi-automatic approach for primitive segmentation, where the user selects the primitive type and one point of the primitive. The primitive is initialized by fitting it into a neighborhood of the point selected by the user. While robust, this approach is not suitable for fully automatic segmentation. Li *et al.* [84] perform a post-processing of detected primitives. They assume that man-made structures were scanned that exhibit certain regularities (parallel planes, repeating structures), and globally optimize detected primitives to adhere to those regularities. While the results are very good for reverse engineering, they report runtimes of 3-10 minutes. Ioannou *et al.* [73] use a multi-scale *Difference of Normals* (DoN) filter that estimates normals using local neighborhoods of different sizes, and computes by how much those estimates differ. Regions are then clustered based on the magnitude of those differences. The method can be highly parallelized and shows good results for scene understanding of outdoor scenes. However, it does not immediately segment the primitives we desire. Willis and Zhou [151] propose a segmentation based on finding closed ridges that segment a surface into two disjoint parts. Their approach requires a connected surface, which is not always available.

## 7.2 Method

In the following, we first present the overall detection pipeline for primitives. The details of each step are then described in the subsequent sections.

### 7.2.1 Detection Pipeline

Our detection of primitive shapes follows a *pipeline* approach. Each step of the pipeline processes a set of candidates and modifies or filters them. We found that the pipeline shown in Fig. 7.1, which is similar to the detection pipeline for the

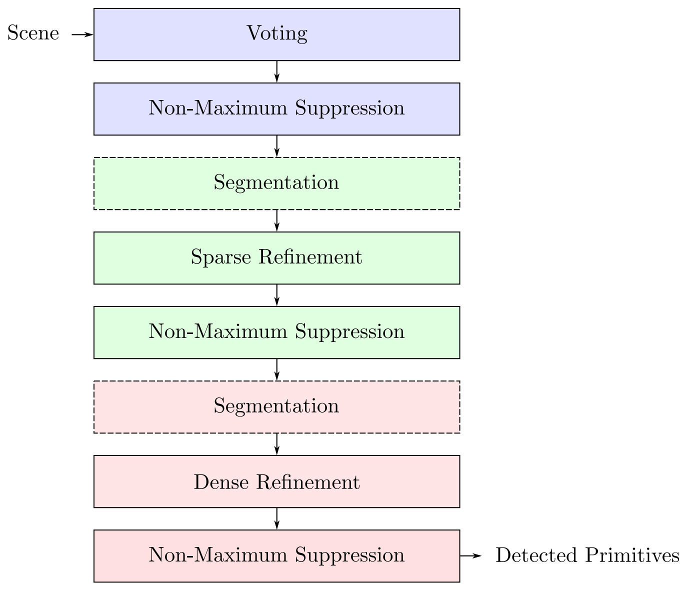


Figure 7.1: Detection pipeline for all primitives. Given some scene, the local voting scheme is used to generate a set of primitive candidates. The resulting candidates are then refined twice using a variant of ICP, first using a sparsely sampled scene, then using the full scene. In between, duplicate candidates are removed.

rigid object detection (Sec. 5.3.9), gives good results. The voting step generates an initial set of hypotheses, which are first refined using a sparsely sampled scene and then refined more accurately using all scene points. After each step, a non-maximum suppression is performed, which removes similar hypotheses, keeping only the one with the locally highest score. The details of the voting, the refinement and the non-maximum suppression vary based on the primitive type.

The two-step refinement, first sparse, then dense, is used with the same motivation as explained in Sec. 4.5. A sparse cloud is enough to refine the candidate with a good accuracy, but is significantly faster than when using the full point cloud. Since the candidate is then more accurate, fewer iterations of the more expensive full-cloud refinement are required.

After each step, the candidates are *scored* by some metric. For the voting scheme, the number of votes cast for each candidate is used as score. After the refinement steps, a more accurate score is computed that depends on the particular primitive and is explained below. For performance, candidates with low scores can be pruned to avoid the computational costs of processing them in later stages. The score is also used to decide which candidates to keep in the non-maximum suppression steps.

For the voting scheme, the point cloud is subsampled to speed up the voting and to remove the bias towards more densely sampled parts of the scene. The sampling distance is computed relative to the approximate size of the primitive. This requires the user to give some approximation for the length of cylinders and for the size of planes.

**Non-Maximum Suppression** The non-maximum suppression removes candidates that are similar to other candidates by keeping only the one with the highest score. If multiple reference points are selected from a primitive's surface, the voting will generate a candidate for the same primitive for each of them. Remove such duplicates in the early steps of the pipeline improves the overall performance, as less candidates need to be processed. After the last step, it avoids returning more than one result for each primitive.

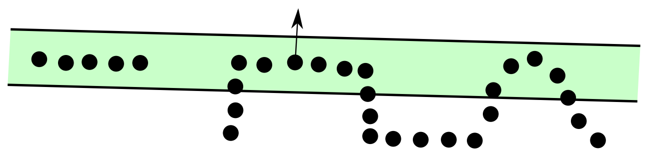
The non-maximum suppression is performed by first sorting all candidates by their score. The candidates are then processed from best to worst; for each one, all similar candidates that have a lower score are removed. To improve performance, a primitive-dependent indexing is used to speed up the search for potentially similar candidates. For spheres, a spatial index of the center is used. For cylinders and planes, an index over the direction of the main axis and the normal vector, respectively, is used. The following pseudocode illustrates this.

```
Input: Set of candidates $c_i \in C$, each with a score $s(c_i)$

Sort $C$ by score
Generate primitive-dependent index $I$ over $C$
for $i$ from 1 to $|C|$ do:
  for each $c_j$ that is similar to $c_i$ (found using $I$) and where $j > i$
    remove $c_j$ from $C$

Output: Set of sorted and de-duplicated candidates $C$
```

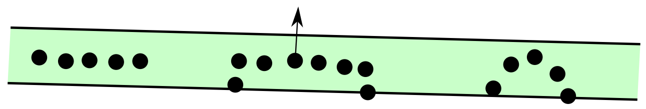
**Segmentation** For planes and cylinders, which are potentially unbounded, an additional and optional segmentation step can be performed prior to the refinement. The segmentation is responsible for removing far-away parts of the scene which would lie on the primitive, if it was unbounded, but which one would not consider to be on the primitive in practice. The segmentation is required since the voting does not give exact bounding information about the primitive. Fig. 7.2 illustrates the segmentation.

First, all points that are on or close to the primitive are detected. The thresholds are determined from the expected noise in the point positions and normal directions, as well as – for the sparse refinement – the expected inaccuracy due to the voting.

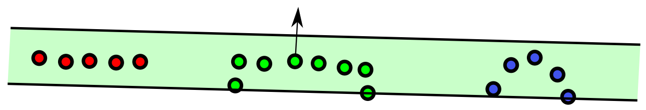
Second, the resulting set of points is decomposed into connected components. The allowed minimum gap between two point sets for them to become separated is a user-defined parameter. It should be larger than the expected point density. Only the component that contains the reference point from the voting scheme is used for the refinement of the primitive.



(a)



(b)



(c)

Figure 7.2: Illustration of the segmentation. (a) A scene point cloud (black points) and a plane candidate. The green region includes all scene points that are on the plane, assuming a certain level of noise. (b) The potential points on the plane are selected. (c) The selected points are partitioned into connecting components. Only the component that contains the original reference point (here the green, central one) is used for the refinement.

### 7.2.2 Local Parameter Space

As we did for rigid 3D objects in Sec. 5.3.3, for the voting scheme, we parametrize the position of each primitive relative to some fixed, oriented scene point using *local coordinates*. However, the properties of primitives allow for the following modifications, which are summarized in Fig. 7.1.

**Removing Redundant Parameters** While $(\mathbf{m}, \alpha)$ is good for parametrizing the transformation of an arbitrary free-form object, it is an overparametrization for geometric primitives. Because of their symmetries, poses of primitives can be described with fewer parameters than poses of free-form objects.

**Planes and spheres** with fixed radius are symmetric w.r.t. rotations around normal vectors, and they look identical from each point on their surface. Their local parameter space thus requires *no* parameters. In other words, a single 3D point and its normal fully defines the plane and the fixed-radius sphere it is on. For **cylinders** with fixed radius, the position of the corresponding point on their surface is irrelevant, however, they are not symmetric w.r.t. rotations around normal vectors. The local parameter space for cylinders with fixed radius is thus one-dimensional and contains only $\alpha$.

The reduced local parameter space has several advantages. First, depending

Table 7.1: Summary of proposed parameter space dimensions.

| | **Number of parameters** | | |
|---|---|---|---|
| **Shape** | **Rigid** | **Local** | **Shape** |
| Free-form | 6 | 3 | 0 |
| Plane | 3 | 0 | 0 |
| Sphere | 3 | 0 | 1 (radius) |
| Cylinder | 4 | 1 | 1 (radius) |

on the shape, it requires no sampling of the model's surface and of the rotation angle, thus avoiding the binning problem. Second, it is more robust due to fewer redundant variables. Finally, it is computationally more efficient, since fewer votes need to be cast and a smaller space needs to be analyzed.

**Shape Parameters** As another modification, the local parameter space can be *extended* by certain non-rigid shape parameters of the object. For example, in certain applications, the radius of spheres and cylinders might be unknown or might vary within a certain predefined range. We thus extend the local parameter space by a non-rigid shape parameter of the primitives, namely their radius. Note that fundamentally, extensions of the local parameter space by additional non-rigid shape parameters – such as the scale of the object – are possible even for free-form objects. However, since the parameter space grows exponentially with the number of parameters, it quickly becomes too large for practical applications. Additional shape parameters are thus especially suited for symmetric objects, for which the local parameter space is already reduced as described above.

The number of votes corresponds to the number of scene points that lie on the target object. For a uniformly sampled scene, when introducing a scaling shape parameter such as the radius of a sphere, this introduces a bias towards larger shapes: Larger shapes have a larger surface area, and more scene points will lie on them, even if a smaller ratio of the shape's surface is visible. To remove this bias, we will normalize the voting space for such shapes by dividing the number of votes of each cell through the influence of the shape parameter on the surface. For example, for spheres, the number of votes for a bin is divided through $r^2$.

In practice, a user-defined range of possible radii $[r_{\text{min}}, r_{\text{max}}]$ is used. For the voting scheme, we sample the radius uniformly in steps of

$$s_{\text{radius}} = r_{\text{max}} / n_{\text{radius}} \tag{7.1}$$

### 7.2.3 Point Pairs

For the detection scheme in Sec. 5, point pairs are extracted from the scene and matched against the model using a hash table. That approach, while fast,

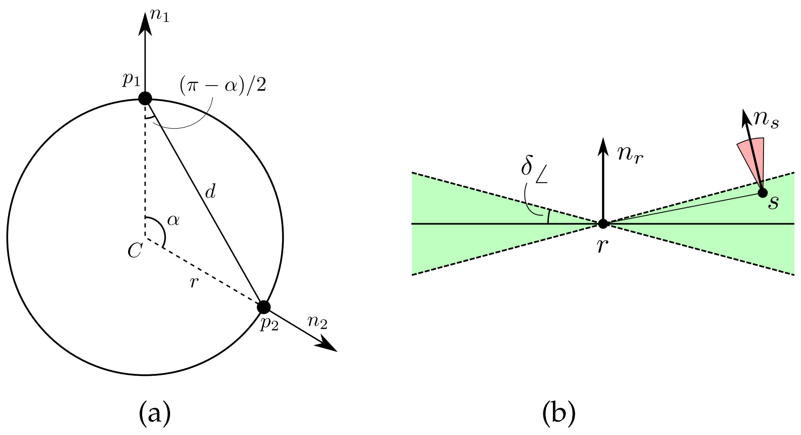


Figure 7.3: (a) Point pair feature for spheres. Given a sphere with center $\mathbf{C}$ and radius $r$, the normals of two points $\mathbf{p}_1$ and $\mathbf{p}_2$ on the sphere form the angle $\alpha$. (b) Illustration of the influence of angular noise on point pairs of a plane. Given a reference point $\mathbf{r}$ and its normal $\mathbf{n}_r$, a scene point $\mathbf{s}$ is considered to be ″on“ the plane defined by $\mathbf{r}$ and $\mathbf{n}_r$, if $|\measuredangle(\mathbf{s} - \mathbf{r}, \mathbf{n}_r) - \pi/2| < \delta_\measuredangle$, *i.e.*, if it is on a plane that is within the tolerance $\delta_\measuredangle$ of the normal vector. The direction of the normal $\mathbf{n}_s$ of $\mathbf{s}$ must also be within that tolerance.

introduces several issues: Since the feature vectors are discretized for the hashing, mismatches can happen at the sampling boundaries; an offline phase is required to build the hash table; a static hash table does not allow shape parameters, such as a radius, or shapes with a-priori unknown size, such as large planes or long cylinders; and finally, it does not allow an explicit model of the expected noise in point positions and normal directions.

In this section, we instead use the implicit nature of primitives to match point pairs and to extract possible shape parameters. Such an implicit database has several advantages. First, it does not require an expensive pre-processing phase. Second, depending on the primitive type, it has fewer parameters: For planes and spheres, the model's sampling distance as well as the feature discretization parameters are no longer required. Third, implicit databases allow to model "infinitely" large planes or cylinders. This allows to search for such primitive objects without knowing their size in advance. Finally, the discretization and hashing approach can lead to missed matches for features close to the discretization boundaries. This problem is avoided with an implicit database, making the method more robust. As downside, the implicit database might be computationally more expensive compared to an explicit database.

In the following, $\delta_p \ll \mathrm{diam}(M)$ is the expected approximate noise in the position of the points, and $\delta_\measuredangle \ll \pi/2$ is the expected approximate angular error of the normal vectors. We assume the positional noise to be much smaller than the diameter of the primitive and ignore its influence onto the angular components of the point pairs.

**Plane** Ideally, all point pairs on a plane have parallel normal vectors and an angle of 90° between the difference vector of the two points and the normal vectors. Given two points $\mathbf{p}_1$ and $\mathbf{p}_2$ with normals $\mathbf{n}_1$ and $\mathbf{n}_2$ and their difference vector $\mathbf{d} = \mathbf{p}_2 - \mathbf{p}_1$, the corresponding point pair features are

$$\begin{aligned} \mathbf{F}(\mathbf{p}_1, \mathbf{p}_2) &= (|\mathbf{d}|, \measuredangle(\mathbf{n}_1, \mathbf{d}), \measuredangle(\mathbf{n}_2, \mathbf{d}), \measuredangle(\mathbf{n}_1, \mathbf{n}_2)) && (7.2)\\ &= (|\mathbf{d}|, \pi/2, \pi/2, 0) && (7.3) \end{aligned}$$

Since planes are infinitely large, $|\mathbf{d}|$ can take any non-negative value.

In practice, both the point coordinates and the directions of the normal vectors will be affected by noise. As illustrated in Fig. 7.3 (b), all point pair features on a plane can then be described as

$$\begin{aligned} \mathbf{F}(\mathbf{p}_1, \mathbf{p}_2) \in \mathbb{R} \times &[\pi/2 - \delta_\measuredangle, \pi/2 + \delta_\measuredangle] \times \\ &[\pi/2 - \delta_\measuredangle, \pi/2 + \delta_\measuredangle] \times \\ &[0, 2\delta_\measuredangle] && (7.4) \end{aligned}$$

For performance reasons, we evaluate (7.4) using the dot product instead of computing the angles:

$$\begin{aligned} cos(\pi/2 + \delta_\measuredangle) &\le \mathbf{n}_1 \cdot \mathbf{d}/|\mathbf{d}| \le cos(\pi/2 - \delta_\measuredangle) && (7.5)\\ cos(\pi/2 + \delta_\measuredangle) &\le \mathbf{n}_2 \cdot \mathbf{d}/|\mathbf{d}| \le cos(\pi/2 - \delta_\measuredangle) && (7.6)\\ cos(2\delta_\measuredangle) &\le \mathbf{n}_1 \cdot \mathbf{n}_2 && (7.7) \end{aligned}$$

We thus check if a scene point pair is on a plane by evaluating (7.5)–(7.7).

**Sphere** As with planes, the point pair database for spheres can be defined implicitly. As visualized in Fig. 7.3 (a), given a sphere with radius $r$, the point pairs can be described based on the angle $\alpha$ formed by the sphere's center and the two points:

$$\mathbf{F}(\mathbf{p}_1, \mathbf{p}_2) = (2r \sin(\alpha/2), (\pi - \alpha)/2, (\pi + \alpha)/2, \alpha) \qquad (7.8)$$

Given a scene point pair, we want to evaluate if it lies on a sphere, and if yes, what possible radii that sphere might have. We write $\alpha$ and $|\mathbf{d}|$ for the measured values from the feature components $\mathbf{F}_1$ and $\mathbf{F}_4$, and $\overline{\alpha}$ and $\overline{|\mathbf{d}|}$ for their underlying exact ground truth values. Since both normals might be off by an angle of up to $\delta_\measuredangle$, we know that

$$\begin{aligned} \overline{\alpha} &\in [\max(0, \alpha - 2\delta_n), \alpha + 2\delta_n] && (7.9)\\ \overline{|\mathbf{d}|} &\in [\max(0, |\mathbf{d}| - \delta_p), |\mathbf{d}| + \delta_p] && (7.10) \end{aligned}$$

The range of possible radii is therefore

$$r \in \left[ \frac{|\mathbf{d}| - \delta_p}{2\sin((\alpha + 2\delta_n)/2)}, \frac{|\mathbf{d}| + \delta_p}{2\sin(\max(0, \alpha - 2\delta_n)/2)} \right] \qquad (7.11)$$

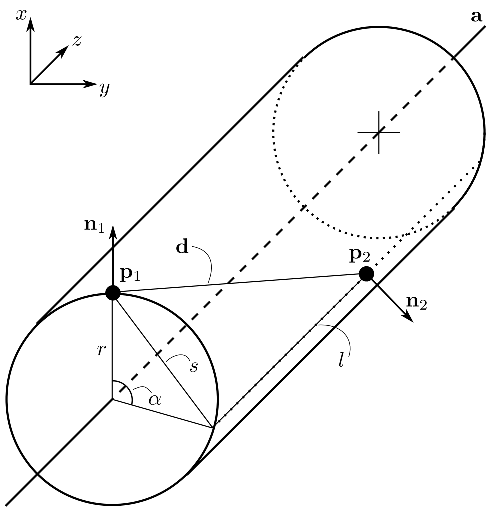


Figure 7.4: Point pair feature for cylinders.

Note that for $\alpha < 2\delta_n$, the upper boundary for $r$ can be infinity. This corresponds to two points with almost parallel normal vectors which could lie on a sphere with an infinitely large radius.

The remaining two components $\mathbf{F}_2$ and $\mathbf{F}_3$ of the feature, the angles between the normal vectors and the difference vector, are not used for the computation above. However, their value is checked against the possible range of values and the point pair is discarded as not being on a sphere if one of those two checks fails.

**Cylinder** While an implicit model for point pairs on a cylinder is possible, such a model would be rather complex and it is computationally expensive to extract the voting parameters $\alpha$ and $r$ from a given point pair. Instead, we pre-create a model that maps given point pairs to the corresponding voting parameters. Note that we only detect and refine the walls of the cylinders, not the caps.

As depicted in Fig. 7.4, let $r$ be the cylinder's radius, $l$ the distance of the two points along the cylinder's main axis, $\alpha$ the angle formed by the two points and the main axis when projected along the cylinder's axis (similar to Fig. 7.3 (a)) and $s$ the distance of the two points if projected along the cylinder's axis. Without loss of generality, we assume that the axis is $\mathbf{a} = (0,0,1)^T$ and that $\mathbf{n}_1 = (1,0,0)^T$. The feature vector is then

$$\begin{aligned}\mathbf{F}(\mathbf{p}_1,\mathbf{p}_2) &= (|\mathbf{d}|, \measuredangle(\mathbf{n}_1,\mathbf{d}), \pi - \mathbf{F}_2, \alpha)\\ &= (\sqrt{s^2+l^2},\\ &\quad\ \measuredangle((1,0,0)^T, (r(1-\cos\alpha), r\sin\alpha, l)^T),\\ &\quad\ \pi - \mathbf{F}_2, \alpha)\end{aligned} \tag{7.12}$$

Note that the three angles of the features do not depend on the radius, while the length component scales linearly with the radius. On a cylinder with radius

$r$, two points will form the point pair $\mathbf{F}^r(\mathbf{p}_1, \mathbf{p}_2)$. Scaling the complete system by $1/r$ leads to the feature

$$\mathbf{F}^1(\mathbf{p}_1, \mathbf{p}_2) = (\mathbf{F}_1^r/r, \mathbf{F}_2^r, \mathbf{F}_3^r, \mathbf{F}_4^r) \tag{7.13}$$

We pre-compute the list of possible features, using a cylinder with radius 1 and some maximum length. Due to the symmetries of a cylinder, we can fix a single point $\mathbf{p}_1$ from that cylinder and pair it with all its other points $\mathbf{p}_2$. The complexity of the cylinder model generation is then $\mathcal{O}(N)$, compared to $\mathcal{O}(N^2)$ for a non-symmetric, free-form model, where every point is paired with every other point. In the online phase, those features are matched against the observed features using only the scale-invariant angular components $\mathbf{F}_2$, $\mathbf{F}_3$ and $\mathbf{F}_4$. For each matching feature, we compute the radius as

$$r = \mathbf{F}_1^r/\mathbf{F}_1^1 \tag{7.14}$$

The matching point pair is discarded if the radius is outside the range of allowed radii.

Since scaling does not change angles, we can pre-compute the rotation angles $\alpha_m$ for the voting scheme as described in Sec. 5.3.5.

While cylinders can have arbitrary lengths, we must decide for a finite length when sampling the prototype cylinder of radius 1. However, note that for very large ratios $l/r$ (*i.e.*, for points very far away from each other w.r.t. the radius of the cylinder), the feature converges towards

$$\mathbf{F}(\mathbf{p}_1, \mathbf{p}_2) = (|\mathbf{p}_1 - \mathbf{p}_2|, \pi/2, \pi/2, \alpha) \tag{7.15}$$

Since we use only the angles for feature lookup, all point pairs for which $|\mathbf{p}_1 - \mathbf{p}_2|$ exceeds a certain threshold will therefore fall into the same bins. We thus add a set of special bins, for which $\mathbf{F}_2 = \mathbf{F}_3 = \pi/2$ and $\mathbf{F}_4 \in [0, \pi]$. If a scene feature falls into one of those bins, a vote is cast for all allowed radii.

### 7.2.4 Refinement

**Framework** The results of the voting scheme will inherently be slightly incorrect because of noise in the input data and sampling of the parameter and feature spaces. Similar to Sec. 4, we use an iterative re-weighted least squares approach to refine the candidates, *i.e.*, to minimize the weighted sum of squared distances from the primitive to the observed scene data. We mostly follow the framework of Sec. 4, except for the following differences:

- Instead of finding the closest model point for each scene point, the distances between scene and primitive are computed explicitly using the primitive's corresponding formula,

- the update of the primitive's parameters are parametrized differently to avoid overparametrization due to symmetries, and
- for planes, no inner iteration is required since the optimization can be solved directly.

Formally, following Sec. 4.2.1 and (4.1), we want to minimize

$$\begin{aligned} E(p) &= \sum_{\mathbf{s}\in S} w_{\mathbf{s}} d_{\text{primitive}}(p,\mathbf{s})^2 && (7.16) \\ &= \sum_{\mathbf{s}\in S} E_{\mathbf{s}}(p)^2 && (7.17) \end{aligned}$$

where $p$ are the primitive's parameters and $d_{\text{primitive}}(p,\mathbf{s})$ is the distance between scene point $\mathbf{s} \in S$ and the surface of the primitive. In each step of the iteration, we find an update $d$ that minimizes $E(d+p)$. Sec. 4.2.1 outlines how this is re-formulated and solved using an iterative Gauss-Newton method. Note that the outer iteration that adapts the weights $w_{\mathbf{s}}$ remains.

**Parametrizations and Updates** The parametrizations $p$ that represent a particular primitive are chosen such that they are smooth over the complete parameter space, thus aiding the numerical robustness. For cylinders, this is an overparametrization. Note that this is different from the parametrization of the update step of the iterative least squares, where we will use a parametrization that is not an overparametrization, thus avoiding null spaces in the solver, and that is locally smooth, *i.e.*, around the 0 update, making the update more robust.

**Spheres** are parametrized as $p = (\mathbf{c}, r)$, using their center $\mathbf{c}$ and their radius $r > 0$. This leads to the distance function

$$d_{\text{sphere}}(p,\mathbf{s}) = |\mathbf{s} - \mathbf{c}| - r \tag{7.18}$$

The update is parametrized directly as $(\delta_{\mathbf{c}}, \delta_r)$.

**Cylinders** require a somewhat more complicated parametrization to avoid discontinuities in the parameter space. Each cylinder is parametrized as $p = (M, r)$, $M \in \text{SE}(3)$, $r > 0$. $M$ is a rigid 3D transformation that maps the cylinder's main axis onto the $z$-axis. To be numerically more robust, we choose the initial $M$ such that the cylinder is close to or encloses the origin. The distance of a point $\mathbf{s}$ from the cylinder is then

$$d_{\text{cylinder}}(p,\mathbf{s}) = |M\mathbf{s} - M\mathbf{s} \cdot (0,0,1)^T| - r \tag{7.19}$$

Note that $(M, r)$ is an overparametrization of the cylinder. To avoid overparametrization of the update, following Sec. 4.2.1, we parametrize the update $\delta$ as

$$\delta = (\delta_r, \delta_{tx}, \delta_{ty}, \delta_{rx}, \delta_{ry}) \tag{7.20}$$

where $\delta_r$ encodes the change in the cylinder's radius, $\delta_{tx}$ and $\delta_{ty}$ encode the change in the cylinder's position orthogonal to its main axis, and $\delta_{rx}$ and $\delta_{ry}$ encode the change in the cylinder's main axis by tilting it. The tilt of the main axis is performed by interpreting $(\delta_{rx}, \delta_{ry}, 0)^T$ as Rodruiges parameters of the corresponding rotation, which effectively tilts the $z$-axis. This parametrization removes both symmetric transformations, the rotation around the $z$-axis and translation along the $z$-axis, from the parametrization. The complete update $T(\delta_{tx}, \delta_{ty}, \delta_{rx}, \delta_{ry})$ is composed of the translation $(\delta_{tx}, \delta_{ty}, 0)^T$ and the described rotation, or

$$T(\delta_{tx}, \delta_{ty}, \delta_{rx}, \delta_{ry})\mathbf{x} = R((\delta_{rx}, \delta_{ry}, 0)^T)\mathbf{x} + (\delta_{tx}, \delta_{ty}, 0)^T \tag{7.21}$$

Given current cylinder parameters $p_k = (M_k, r_k)$ and an update $\delta$, the new parameters $p_{k+1} = (M_{k+1}, r_{k+1})$ are computed as

$$r_{k+1} = r_k + \delta_r \tag{7.22}$$

$$M_{k+1} = T(\delta_{tx}, \delta_{ty}, \delta_{rx}, \delta_{ry})M_k \tag{7.23}$$

For spheres, all partial derivatives are computed analytically. For cylinders, the partial derivatives w.r.t. $\delta_r$, $\delta_{tx}$ and $\delta_{ty}$ are computed analytically, while those w.r.t. $\delta_{rx}$ and $\delta_{ry}$ are computed numerically.

**Planes** are parametrized as $p = (\mathbf{n}, d) \in \mathbb{R}^3 \times \mathbb{R}$ using the normal form

$$d_{\text{plane}}(p, \mathbf{s}) = \mathbf{n} \cdot \mathbf{s} + d \tag{7.24}$$

with the constraint $|n| = 1$. This gives

$$E(p) = \sum_{\mathbf{s} \in S} w_{\mathbf{s}} (\mathbf{n} \cdot \mathbf{s} + d)^2 \to \min \tag{7.25}$$

We can eliminate $d$ by solving $\delta E / \delta d = 0$, giving

$$d = -\mathbf{n} \cdot \left( \frac{1}{N} \sum_{\mathbf{s} \in S} w_{\mathbf{s}} \mathbf{s} \right) = -\mathbf{n} \cdot \overline{\mathbf{s}_w} \tag{7.26}$$

where $\overline{\mathbf{s}_w}$ is the weighted mean of $S$. Substituting (7.26) in (7.25) gives

$$E(p) = \sum_{\mathbf{s} \in S} w_{\mathbf{s}} (\mathbf{n} \cdot \mathbf{s} - \mathbf{n} \cdot \overline{\mathbf{s}_w})^2 \tag{7.27}$$

$$= \sum_{\mathbf{s} \in S} w_{\mathbf{s}} (\mathbf{n} \cdot (\mathbf{s} - \overline{\mathbf{s}_w}))^2 \tag{7.28}$$

$$= \sum_{\mathbf{s} \in S} w_{\mathbf{s}} (\mathbf{n} \cdot \hat{\mathbf{s}})^2 \tag{7.29}$$

where $\hat{\mathbf{s}} = s - \overline{\mathbf{s}_w}$. Intuitively, the scene points are translated such that their weighted mean is zero. The partial derivatives w.r.t. the components of $\mathbf{n}$ are

$$\frac{\delta E}{\delta \mathbf{n}_i} = \sum_{\mathbf{s} \in S} w_{\mathbf{s}} \mathbf{n} \cdot \hat{\mathbf{s}} \hat{\mathbf{s}}_i, \quad i \in \{1, 2, 3\} \tag{7.30}$$

$$= \left( \sum_{\mathbf{s} \in S} w_{\mathbf{s}} \hat{\mathbf{s}} \hat{\mathbf{s}}_i \right) \cdot \mathbf{n} = 0 \tag{7.31}$$

which can be combined into a single equation

$$0 = \left( \sum_{\mathbf{s} \in S} w_{\mathbf{s}} \hat{\mathbf{s}} \hat{\mathbf{s}}^T \right) \mathbf{n} = X \mathbf{n} \tag{7.32}$$

Due to the constraint $|\mathbf{n}| = 1$, we search for a non-trivial solution of (7.32), *i.e.*, a normalized vector from the kernel of $X$. However, due to noise and numerical issues, the rank of $X$ might be 3. We therefore look for an eigenvector of the smallest eigenvalue of $X$ instead. This is equivalent to a PCA of $S$: The normal direction is the dimension into which $S$ extends the least.

Note that contrary to spheres and cylinders, (7.25) is solved directly for planes. However, we still perform the outer iteration that adapts the weights, in order to remove outliers.

## 7.3 Theoretical Comparison with RANSAC

The proposed local voting method can be seen as a hybrid between a full (non-local, with a single parameter space) voting scheme and RANSAC. While RANSAC selects *multiple* random points, enough to fit the target primitive, the proposed method selects only a *single* random point, the reference point. The primitive parameters are deduced from that single point using the voting scheme, which is deterministic. By selecting the reference points through uniform sampling of the scene, their selection can be seen as quasi-deterministic, compared to the usually more random selection of the multiple points in RANSAC.

Contrary to RANSAC, the method is thus non-random and has a more deterministic runtime. Additionally, since only a single point on the object must be found, the method is significantly faster when the ratio of inlier points in the scene is low. For example, if $p \in [0, 1]$ is the ratio of scene points on the target object (inliers) and RANSAC requires $n$ points to fit the model, the probability to select $n$ inlier points is $p^n$. The proposed method requires only a single reference point to be selected on the target object, which has thus a probability of $p$.[1]

## 7.4 Experiments

We evaluated the method using an unoptimized, partly parallelized C implementation. All timings were measured on an Intel Core i5 with 3.33 GHz and 16 GB RAM.

[1]Of course, many strategies for RANSAC exist to mitigate this problem, such as selecting points that are close to each other.

### 7.4.1 Refinement

In a first set of experiments, we evaluated the proposed refinement algorithms for all three primitives. The tests were done on synthetic data with known ground truth to evaluate the accuracy and basin of convergence. Depending on the primitive, we varied the number of close-range clutter points (outliers), the noise in the data, the noise in the initial parameters of the primitive and the amount of visibility of the primitive.

To make the measurements somewhat comparable and independent of the overall scale of the data, we measure distance-based values relative to the size of the primitives. Two relative sizes are used: The maximum diameter of the primitive and the radius of spheres and cylinders. In particular, the Gaussian noise $\sigma$, which is applied to the scene point positions, the initialization error and the average distance of points on the ground truth primitive to the refined primitive is given relative to the diameter of the primitive, and the error of radius $r$ (cylinder, sphere) and of the sphere center $c$ is given relative to the radius of the corresponding primitive.

The parameters for all tests, except for the correspondingly varied parameter, were a noise of $\sigma = 0.05$, a relative initialization error of 0.03, and a scene size of $\approx 7.000$ points for 100% visibility (and correspondingly fewer points for less visibility).

Fig. 7.5 shows the results for the refinement of spheres, Fig. 7.6 the results for the cylinder, and Fig. 7.9 the results for planes. Fig. 7.7 and 7.8 show quantitative and qualitative data of an example run with $\sigma = 0.045$, initial error 0.05, no clutter and $\approx 10.000$ scene points. Note the fast initial convergence, where a good approximation of the cylinder is already found in only two steps.

Overall, the refinement of all three primitives is highly robust against close-range clutter and copes well even with larger amounts of noise. The basin of convergence is well within the expected inaccuracy of the voting scheme, making the refinement a good extension of the proposed voting scheme. For spheres and cylinders that are only barely visible (10% visibility or less), the refinement becomes less stable, especially in presence of many outliers or noise. However, those situations represent an ill-posed problem, as the remaining part of the primitives becomes more and more planar, making a robust estimation of the radius difficult. Note that we found that the estimated cylinder radius slightly exceeds the ground truth radius. This is due to a characteristic of the Gaussian noise, which – as the cylinder's surface is convex – on average moves more points away from the cylinder's axis than towards it. Though Gaussian noise is a somewhat simplified model for sensor noise, one might have to compensate for this effect.

Table 7.2: Results on the SegComp ABW Dataset [68] (Results from [105] and [58].)

| approach | correct | over | under | missed | noise |
|---|---|---|---|---|---|
| USF | 12.7 (83.5%) | 0.2 | 0.1 | 2.1 | 1.2 |
| WSU | 9.7 (63.8%) | 0.5 | 0.2 | 4.5 | 2.2 |
| UB | 12.8 (84.2%) | 0.5 | 0.1 | 1.7 | 2.1 |
| UE | 13.4 (88.1%) | 0.4 | 0.2 | 1.1 | 0.8 |
| OU | 9.8 (64.4%) | 0.2 | 0.4 | 4.4 | 3.2 |
| PPU | 6.8 (44.7%) | 0.1 | 2.1 | 3.4 | 2.0 |
| UA | 4.9 (32.2%) | 0.3 | 2.2 | 3.6 | 3.2 |
| UFPR | 13.0 (85.5%) | 0.5 | 0.1 | 1.6 | 1.4 |
| MRPS | 11.1 (73.0%) | 0.2 | 0.7 | 2.2 | 0.8 |
| ours | 12.3 (80.7%) | 0.2 | 0.8 | 2.6 | 0.0 |

### 7.4.2 Detection - Quantitative

**ABW SegComp Dataset** We evaluated the plane detector on the SegComp ABW Dataset [68], a database of 30 scenes taken with a laser scanner, and with known ground truth. Table 7.2 shows the results of our detector, compared to prior art. As in the original evaluation, *over* is the average number of oversegmentations (more than one detected primitive for a single ground truth primitive), *under* is the average number of undersegmentation (more than one ground truth primitives for a single detected primitive), *missed* is the average number of undetected primitives and *noise* is the average number of false positives.

Overall, our method was able to detect almost all well-behaving planes. Note that our approach has zero noise, indicating that all of our detected planes corresponded to a ground truth plane (no false positives). The missed planes had a very high inclination w.r.t. to the camera, and sometimes were only a few pixels large, making them very difficult to detect.

**Synthetic Dataset** We also performed an evaluation of the primitive detection on some 1000 artificial scenes with known ground truth and added Gaussian noise. Each scene contained 1-10 primitives, some partially occluding each other. Fig. 7.10 shows the resulting detection rate. All well-behaving primitives that were not too much occluded or tilted against the camera were detected.

### 7.4.3 Detection - Qualitative

We performed a large number of qualitative experiments on real-world data. Scenes were acquired with different sensors, including time-of-flight sensors, Kinect-like structured light sensors and stereo setups. Some of the results are depicted in Figs. 7.11, 7.12, 7.13, 7.14 and 7.15.

While cylinders are detected robustly, the determination of the exact length of the cylinders can be tricky. In some cases, the cylinder is assumed to be slightly longer than it should be if its end is touching a different object. Other than that, we found that all primitives were detected robustly, despite occlusions, clutter and noise.

## 7.5 Conclusion

This chapter introduced a detection scheme for symmetric 3D shapes – cylinders, spheres and planes – in 3D point clouds. For this, the parameter space of the voting scheme of Sec. 5 was modified to take the the symmetries into account: Redundant parameters were pruned and the scale – or radius – was added as shape parameter. Additionally, for spheres and planes, the feature matching is performed implicit instead of using an explicit feature database. Finally, the refinement scheme of Sec. 4 was modified to take the implicit shapes into account and, for cylinders and spheres, to also optimize the radius of the shape. The experiments show that the method is robust, fast, and accurate and detects all three object classes in challenging real-world scenes.

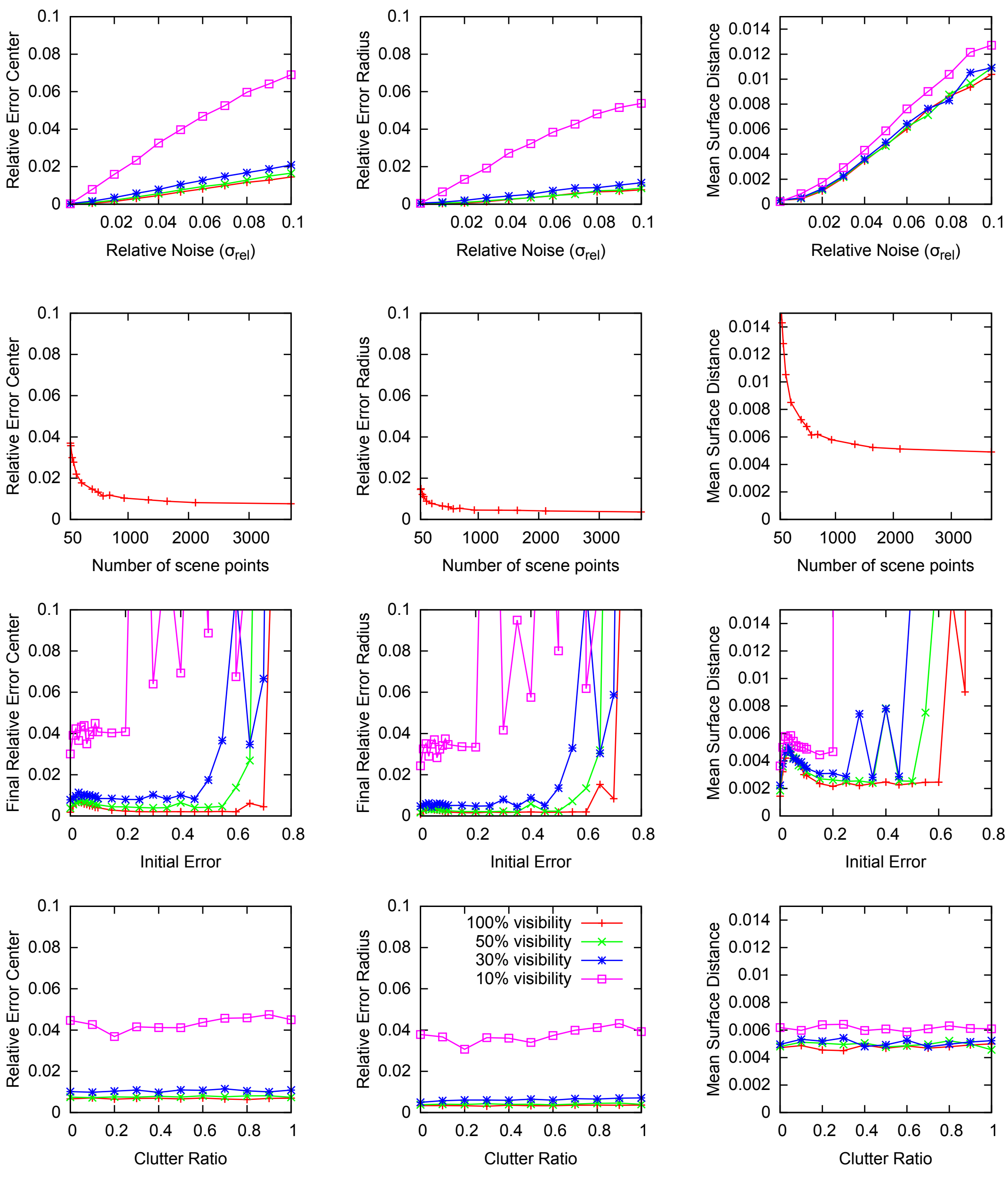


Figure 7.5: Accuracy of the sphere refinement on synthetic data. Each graph shows results aggregated over several thousand runs. More details and discussion can be found in the text. **From left to right:** Relative center error $|\mathbf{c}_{\text{refined}} - \mathbf{c}_{\text{gt}}|/r_{\text{gt}}$, relative radius error $|r_{\text{refined}} - r_{\text{gt}}|/r_{\text{gt}}$, and mean relative distance of points on the ground truth sphere to points of the refined sphere. **From top to bottom:** Gaussian noise on scene points, varied scene density (at 50% visibility), varied initialization error (basin of convergence), varied clutter point ratio (= nr. clutter points/nr. inlier points).

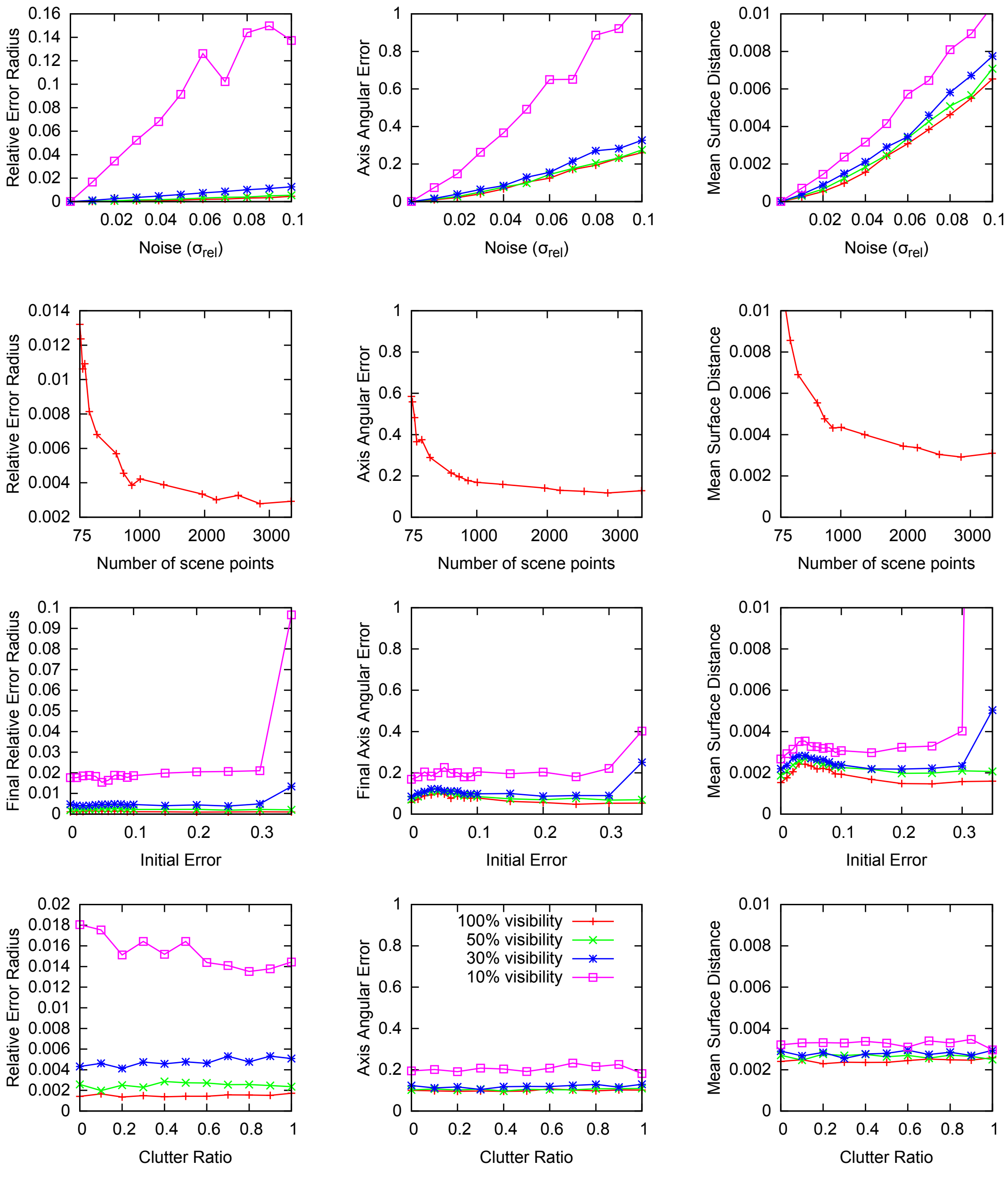


Figure 7.6: Accuracy of the cylinder refinement on synthetic data. Each graph shows results aggregated over several thousand runs. More details and discussion can be found in the text. **From left to right:** Relative radius error $|r_{\text{refined}} - r_{\text{gt}}|/r_{\text{gt}}$, angular error of cylinder axis $\measuredangle(a_{\text{refined}}, a_{\text{gt}})$ in degrees, and mean distance of points on the ground truth cylinder to points of the refined cylinder. **From top to bottom:** Gaussian noise on scene points, varied scene density (at 50% visibility), varied initialization error (basin of convergence), varied clutter point ratio (= nr. clutter points/nr. inlier points).

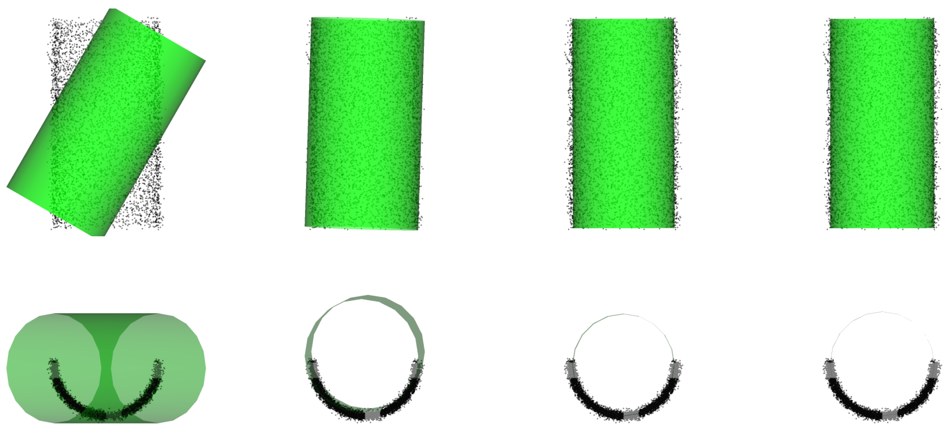


Figure 7.7: Example refinement for a noisy cylinder. Top view (top), frontal view (bottom) for the initialization (left) and the first three iterations. For quantitative details and discussion see Fig. 7.8

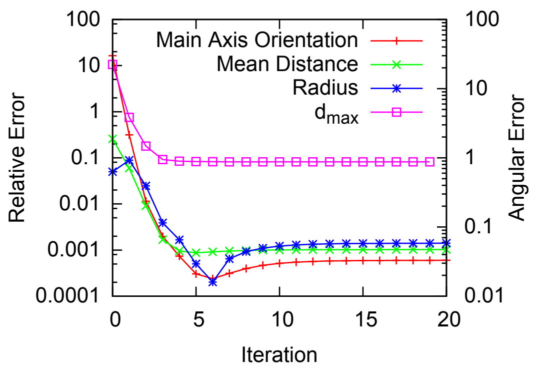


Figure 7.8: Residual error and maximum distance for an exemplary run of the cylinder refinement. The refinement is depicted in Fig. 7.7. Note that the refinement converges in around 10 steps and shows a very fast initial convergence.

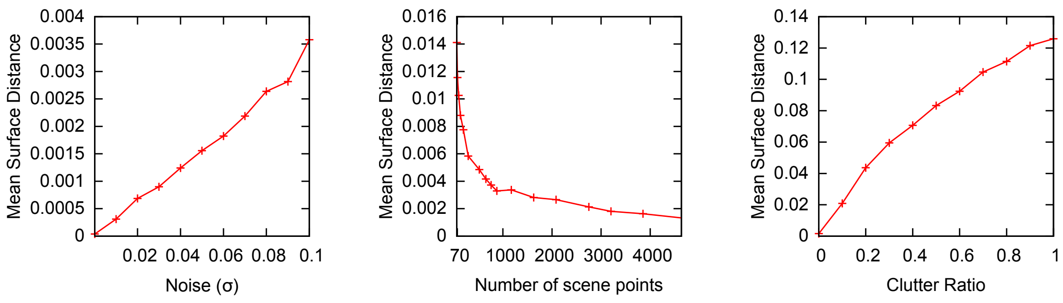


Figure 7.9: Accuracy of the plane refinement on synthetic data. Each graph shows results aggregated over several thousand runs. More details and discussion can be found in the text. **Left:** Accuracy w.r.t. Gaussian noise of the scene points. **Center:** Accuracy w.r.t. the number of scene points. **Right:** Accuracy w.r.t. the clutter ratio (= nr. clutter points/nr. inlier points).

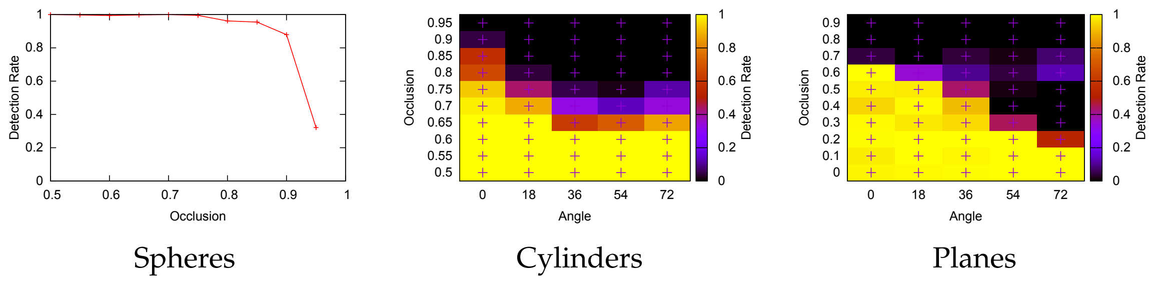


Figure 7.10: Effect of occlusion and tilt w.r.t. the camera on detection rates, averaged over several thousand synthetic scenes. For occluding the primitives, an object was placed between them and the camera such that this occluding object appears in the center of the primitive. Also, at most half each cylinder and sphere was visible. For cylinders and planes, detection rate is also plotted against the tilt of the main axis and normal vector, respectively, w.r.t. the viewing direction.

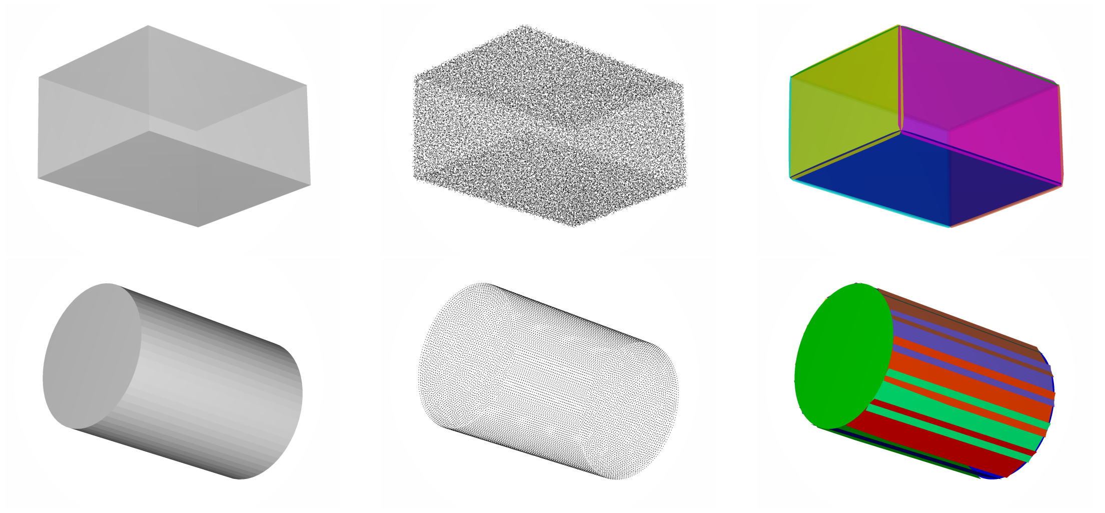


Figure 7.11: Synthetic examples of the plane detection on sampled primitives, a box (top row) with Gaussian noise on the point coordinates, and a cylinder (bottom row). **Left**: Original object. **Center**: Sampled object (which is the input to the plane detector). **Right**: Detected planes.

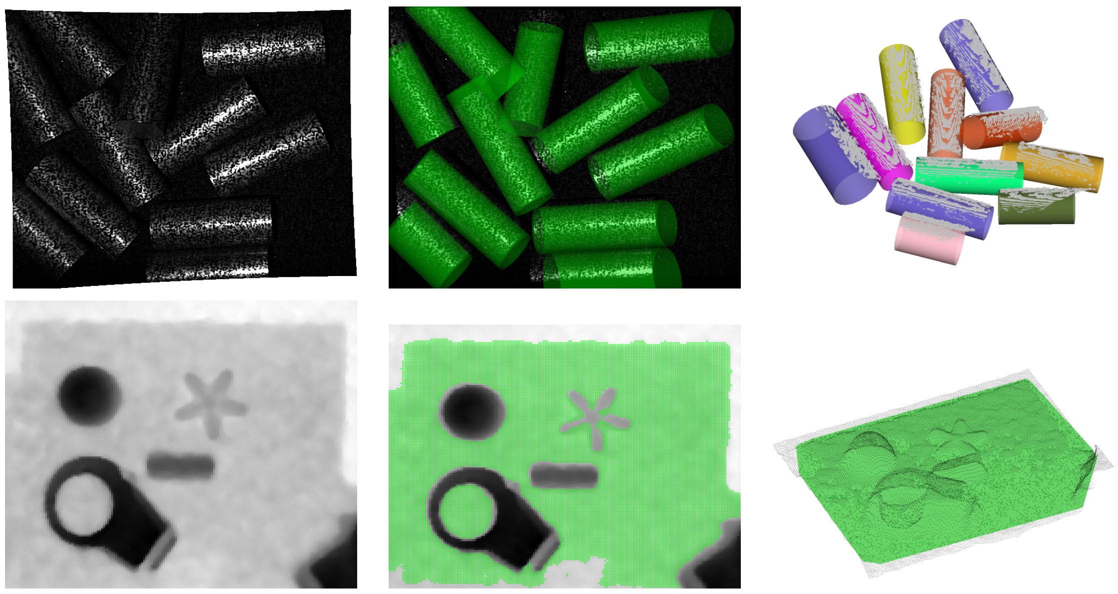


Figure 7.12: Real-world example for detecting cylinders (top) and the background plane (bottom) in an industrial setup. **Top Left**: Original image, showing a projected pattern for stereo reconstruction. **Top Center**: Detected cylinders in the original image. **Top Right**: 3D view of the detected cylinders. Note that all cylinders were detected, however, some with a slightly wrong length. The cylinders were detected in $\approx$ 450ms. **Bottom Left**: Original depth image from a time-of-flight sensor. **Bottom Center**: Points on the detected plane are marked in green. **Bottom Right**: 3D view of the detected plane. Note the high amount of noise from the time-of-flight sensor, which is in the range of $\approx$ 1cm

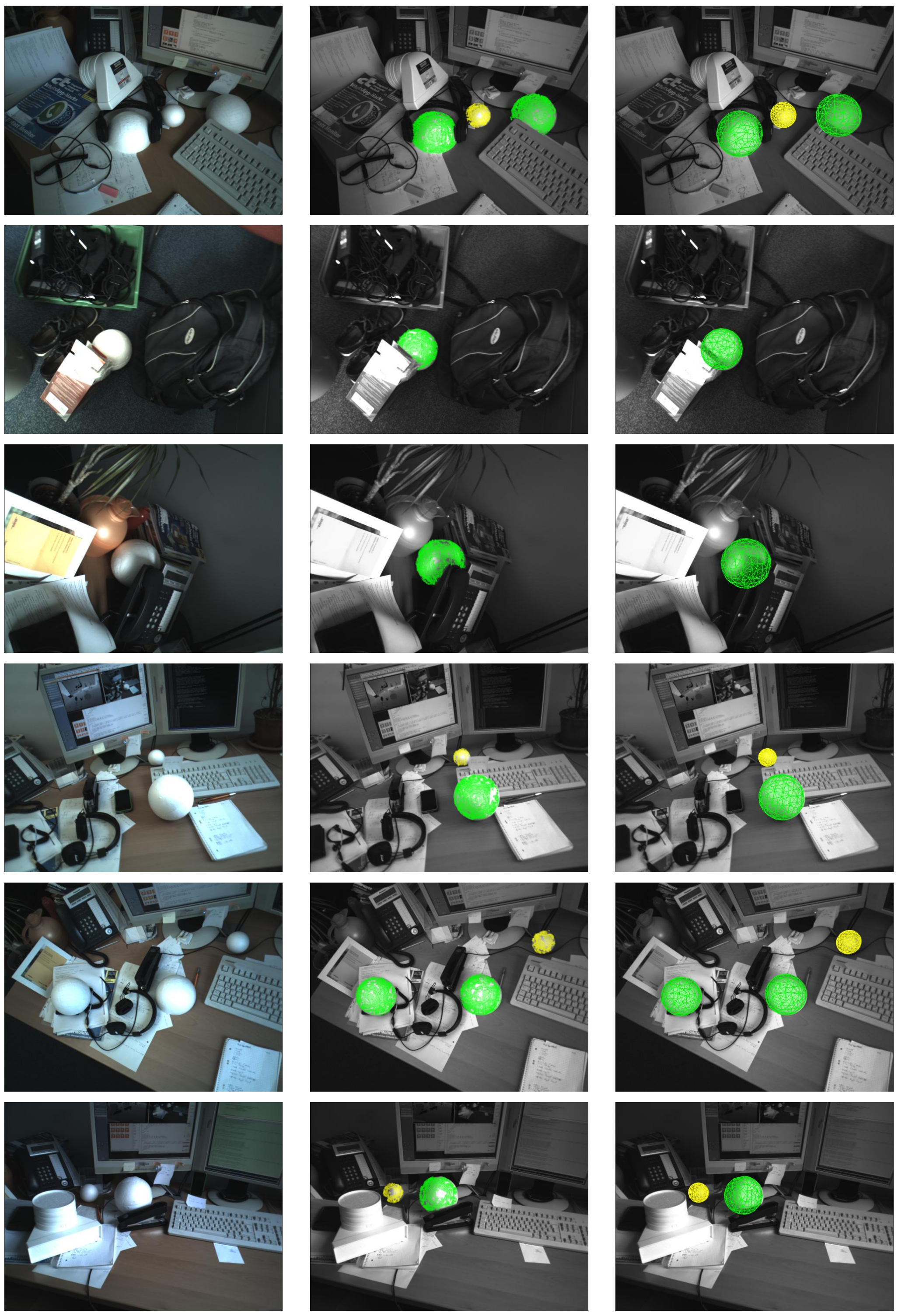


Figure 7.13: Real-world examples for detecting spheres. The examples show heavy clutter, occlusion, multiple instances of the target radius, and spheres of different radius. Note that only the depth image was used for detection and that the RGB image is used solely for visualization. **Left**: Original RGB image. **Center**: Points detected on the sphere(s). **Right**: Detected sphere(s). The average detection time for the scenes was $\approx$ 200ms (including voting and refinement).

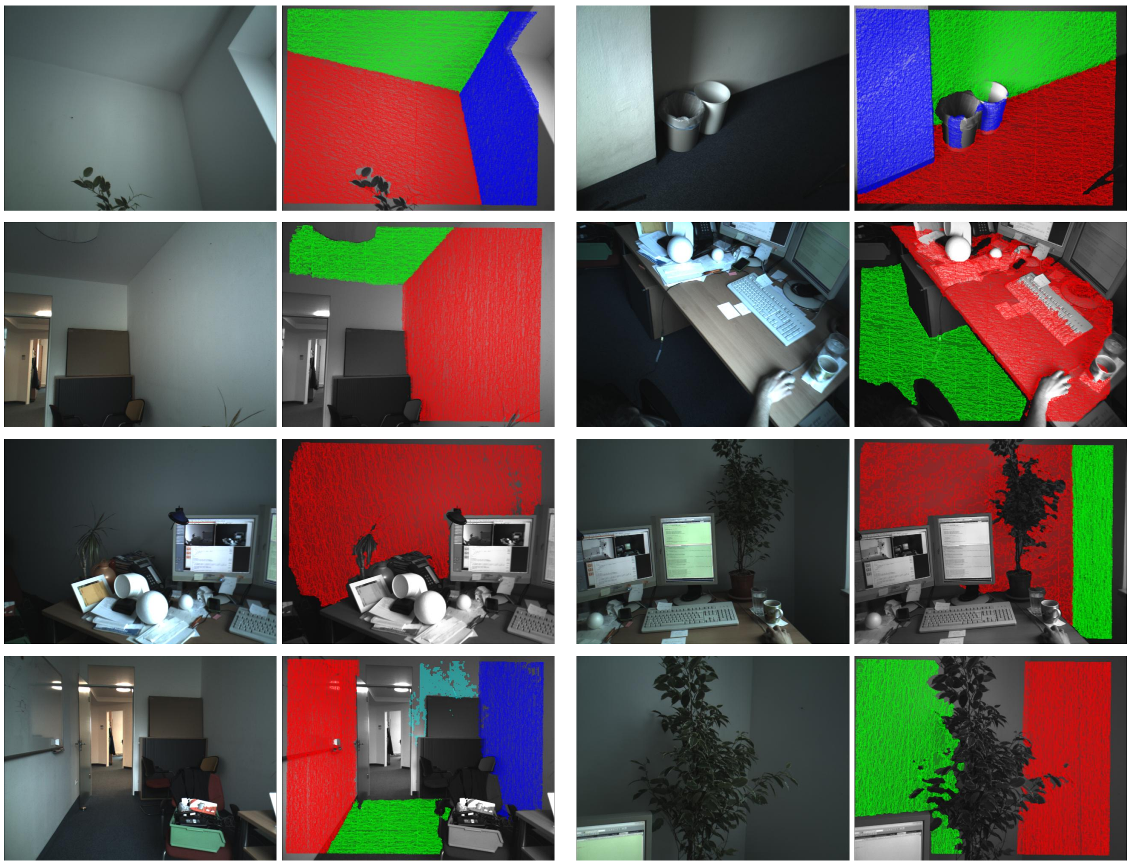


Figure 7.14: Real-world examples for detecting planes. The examples show heavy clutter, occlusion, multiple instances and planes of different size. Note that only the depth image was used for detection and that the RGB image is used solely for visualization. Each pair shows the original RGB image and the detected plane segments. The average detection time for the scenes was $\approx$ 200ms (including voting and refinement).

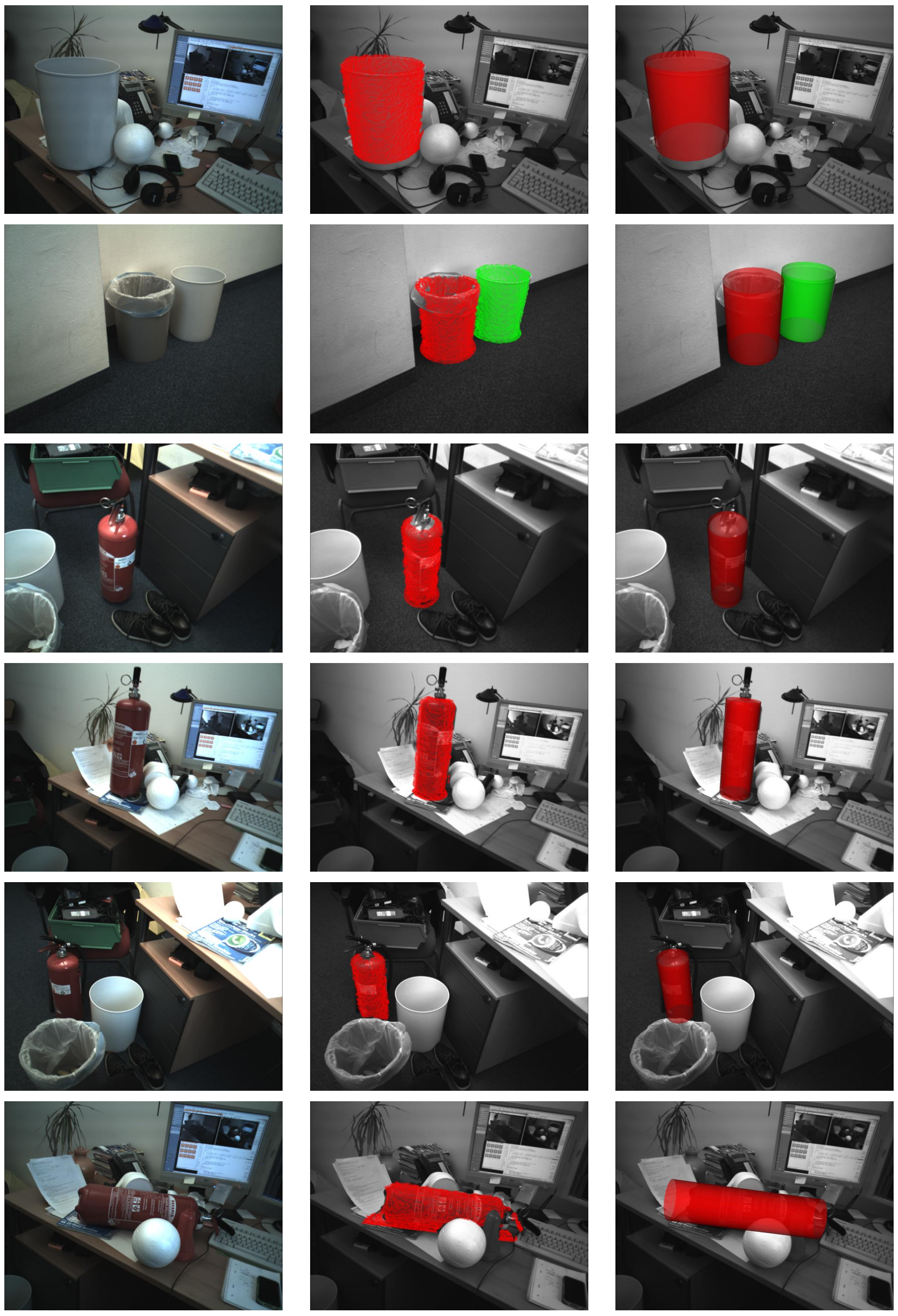


Figure 7.15: Real-world examples for detecting cylinders. The examples show heavy clutter, occlusion, multiple instances of the target radius, and cylinders of different radius. Note that only the depth image was used for detection and that the RGB image is used solely for visualization. **Left**: Original RGB image. **Center**: Points detected on the cylinders(s). **Right**: Detected cylinders(s).

# 8 Deformable 3D Object Detection in 3D Point Clouds

Besides the detection of rigid and primitive objects introduced in the previous chapters, some applications require a detector that can deal with deformable objects, such as parts made from soft plastics, or with object classes that exhibit intra-class variations, such as bakery products or fruits. This chapter introduces a method that detects such deformable objects in 3D point clouds and that returns a consistent set of weighted correspondences between a reference model and the scene.

The proposed method generalizes well over different object classes and requires no explicit deformation model. Most parameters can remain constant over a large range of objects, making the method general and easy to use. In terms of performance, we obtain sub-second runtimes on large scenes.

Parts of this chapter were previously published in [43].

## 8.1 Introduction and Related Work

Like the previous approaches, the method is based on the voting scheme introduced in Sec. 5. However, the method is modified in two ways:

1. The point pair database is extended to include possible deformations of all point pairs. Instead of storing only the point pair features of a single rigid model, the features of possible deformations are stored. We will show how this allows training of deformations without a model, based on a few examples only.
2. The voting is done for all reference points simultaneously, which allows to perform an iterative voting, where the scores from the previous iteration are used as votes in the next voting round. This amplifies consistent correlations. The iterative voting is modeled as a power iteration on a graph adjacency matrix.

The detector is designed for small- to medium-range deformations. In particular, strong articulated motions, such as humans, would be computationally too expensive.

Note that this chapter concentrates on the recovery of approximate, but consistent scene-model-correspondences. Additional model and deformation dependent refinement steps, such as deformable ICP or model fitting, are not performed. We evaluate the approach quantitatively and qualitatively on synthetic and real-world datasets, showing its generality, performance and robustness.

Chui and Rangarajan [32] approached the point correspondence problem in 2D using their TPS-RPM framework that can deal with outliers and uses thin-plate-splines as deformation model. However, their approach was demonstrated on artificial 2D data only. It does not scale well to 3D data with large amounts of clutter due to the worst-case performance of $\mathcal{O}(N^3)$. Anguelov *et al.* [3] solve the correspondence problem in 3D using a joint probabilistic model that preserves local geometry. Their method shows very good results when registering meshes of humans using a deformation model that preserves geodesic distance. While the two preceding methods are able to register deformed variants of point clouds, they are unable to deal with larger amounts of outliers, clutter, noise, or occlusion. They are also limited to a single or few deformation models. Those restrictions make the approaches unsuitable as generic 3D deformable object detectors.

Ruiz-Correa *et al.* [113] proposed a deformable shape detector that uses a symbolic representation of shape components to represent and detect deformable objects. Their method can deal with occlusion and noise, and generalizes well over different deformation models in a "learn by example" way similar to our proposed approach. However, they report runtimes of over 12 minutes, making their method impractical for real-world applications.

The usage of graph matching algorithms in Computer Vision has a long tradition. An extensive overview is given by Conte *et al.* [36]. Graph matching allows a robust localization of deformed objects and is a promising method for such a challenge. While it has been shown extensively to work in 2D applications, its applications in 3D are mostly limited and restricted to artificial perfect-data scenarios (see, for example, Duchenne *et al.* [48]). Berg *et al.* [14] model the assignment problem as an Integer Quadratic Programming (IQP) problem and use a thin-plane spline for post-processing and outlier removal. Leordeanu and Hebert [82] proposed a relaxation of the binary assignment problem, showing that it is orders of magnitudes faster and more robust than IQP. The graph structure in our proposed method is based on their graph, where vertices represent point-to-point assignments, while edges connect geometrically consistent assignments. They also show the connection between the energy optimization and the eigenvector problem of the adjacency matrix. However, they performed no evaluation of the method with deformable 3D models.

Recently, hypergraphs were used for efficient image and point cloud registration. Zass and Shashua [157] proposed to use hypergraphs to model more complex relations between two feature sets. Chertok and Keller [25] build upon that work and show efficient hypergraph matching for 2D images. Duchenne *et al.* [48] use higher-order relations for the graph creation, showing good results in both 2D and 3D. However, they evaluate only on perfect 3D meshes and show no quantitative results in 3D. Also, their creation of the adjacency matrix is expensive and makes their method impractical for real-world applications. Leordeanu *et al.* [83] proposed a new hypergraph matching algorithm, which they use to efficiently register images that contain deformations. Lee *et al.* [81] extend a random walk strategy to hyper-graphs and can include similarity measures of arbitrary orders. They outperform other methods in 2D when matching feature points in 2D images.

Several of the mentioned methods require feature point detectors and were applied on 2D image data only. While robust feature point detectors in 2D are available, 3D data often exhibits too little distinctive geometry for robust salient point or feature point extraction. The method proposed in this chapter thus uses a all-to-all matching that does not require feature point extraction.

Other approaches deal with shape retrieval, *i.e.*, the identification of 3D point clouds or meshes. Passalis *et al.* [109] use a wavelet representation of objects for efficient shape retrieval in large databases. Mahmoudi and Sapiro [86] identify point clouds based on the distribution of several intrinsic measurements on that cloud, such as geodesic distances. While those approaches generalize well to rigid and non-rigid object classes, they require the objects to be segmented, making the approaches unsuitable for scenes with large amounts of clutter.

## 8.2 Method

**Contributions** The method is based in parts on the graph model and matching framework of [82] and [48]. However, we tailor those methods to 3D and to increase their efficiency. In detail, the contributions are:

- The allowed range of deformations is learned based only on observed example deformations. It requires no explicit model and works with only a few examples, making the method easy to use.

- Our vertices represent not only correspondences between two 3D points, but full rigid 3D transformations using the local coordinates introduced in Sec. 5.3.3. This improves the graph's discriminative power.

- The graph is constructed sparsely, using an initial single-round voting to remove a large set of unlikely scene-model-correspondences a priori. This improves the matching performance.

- Finally, we extract the most dominant consistent subgraph using a novel method that is efficient, requires no model, and is virtually parameter free.

**Notation and Input** Both model and scene are subsampled uniformly to avoid any bias from different point densities throughout the point clouds. In practice, we use sampling distances between 3% and 5% of the model's diameter. We denote $\mathbf{m}_i \in M$ for points on the sampled model and $\mathbf{s}_j \in S$ for points on the sampled scene surface. Both point clouds are oriented, i.e., each point has a normal associated with it. The objective is to find a deformed instance of the model in the scene by giving consistent correspondences between scene and model points. Due to occlusion, clutter, and noise, not every scene point has a corresponding model point and vice versa. In the literature, this is sometimes called the *correspondence problem* or the *assignment problem*.

**Overview** In order to find these correspondences, we build a graph $G = (V, E)$, where each vertex $v \in V$ represents a possible correspondence between a scene point and a model point. This chapter sometimes uses *vertex* and *correspondence* interchangeably. An edge $e = (v_1, v_2) \in E$ indicates that some non-rigid transformation exists such that both correspondences $v_1$ and $v_2$ are aligned simultaneously. In other words, vertices that represent consistent correspondences are connected. This graph model is based on [82]. If an instance of the model is present in the scene, the graph's vertices that connect the visible model points to their ground-truth scene points will be connected and form a dense subgraph of $G$. We will extract the most dominant such subgraph and thus recover consistent model-scene-correspondences.

### 8.2.1 Model Generation

**Feature and Database.** As for the rigid matching in Sec. 5, we use oriented point pairs (Sec. 5.3.1), which are discretized and stored in a hash table (Sec. 5.3.2). However, we store all deformed variants of each point pair in the database. To counter the corresponding increase in memory, the discretization steps of angles and distances are increased. Formally, $\mathbf{F}(\mathbf{m}_1, \mathbf{m}_2)$ denotes the point pair feature of two points $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{R}^3$ and $H(\mathbf{F})$ the entries in the hash table $H$.

**Deformation Model.** Real-world object classes exhibit a large variety of different deformations. Modeling all or even most of them is difficult, as is the selection and parametrization of such a model by the user. In order to be independent from any particular deformation model, we learn the range of possible deformations based only on registered examples $M_1, M_2, \dots, M_n$ given by the user. With this approach, no explicit model is required.

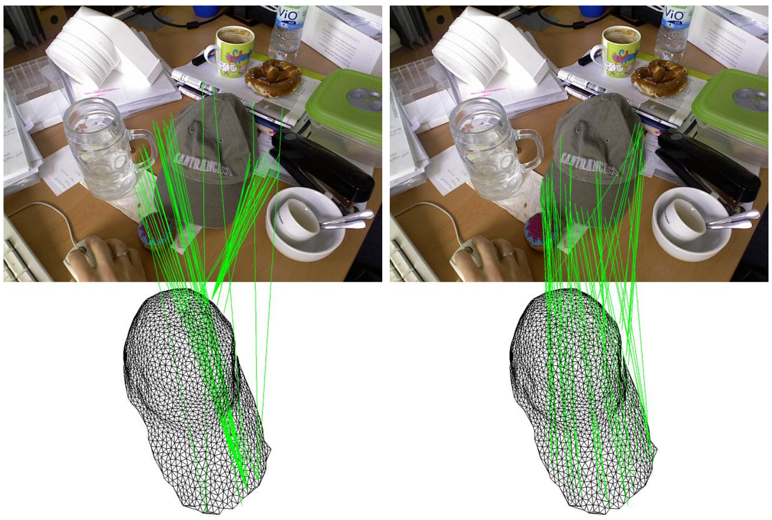


Figure 8.1: Graph matching results when parametrizing vertices with correspondences only (left) or with correspondence and rotation angle (right). The right side has significantly more correct correspondences and leads to a correct match.

We write $\mathbf{m}_i^k \in M_k$ as position of model point $\mathbf{m}_i$ in the deformed example $M_k$. For each pair $(\mathbf{m}_i, \mathbf{m}_j) \in M^2$, we first collect all its deformations

$$D(\mathbf{m}_i, \mathbf{m}_j) = \{(\mathbf{m}_i^k, \mathbf{m}_j^k) : k = 1, \dots n\} \tag{8.1}$$

from the provided examples. We then form all convex combinations between two features in $D$, and add all those combinations to the database. Note that the discretization of the feature vectors adds a small range of possible deformations, since variations that do not change the discretized value do not affect the value retrieved from the hash table.

Note that for large-scale deformations, it might be of advantage to learn the manifold on which each point pair is moving, rather than taking the convex hull of the point's locations. However, this was not further explored in this work.

### 8.2.2 Vertex Parametrization

**Motivation** Our graph models correspondences between model and scene points. In 2D, a single point-to-point correspondence completely captures a rigid motion, assuming that normal vectors or gradients are available. In 3D, however, a single correspondence misses one degree of freedom: After aligning a scene and a model point as well as their normal vectors, one can still rotate around the normal vector. Using correspondences only is thus an underparametrization of an underlying rigid motion. For graph matching, this has the effect of aggregating vertices and thus probably introducing undesired cliques, making it more difficult to extract the correct correspondences.

**Solution** To mitigate this effect, we explicitly include the rotation around the normal in the vertex parametrization. Each vertex in the graph then represents not only two corresponding points $\mathbf{s}$, $\mathbf{m}$, but also a rotation angle $\alpha$ around the normal vector. $(\mathbf{m}, \alpha)$ are also called the *local parameters* w.r.t. $\mathbf{s}$ (compare Sec. 5.3.3). Together with the normals, these parameters completely parametrize a rigid transformation $T$. Formally, we follow Sec. 5.3.3 and define $T$ as

$$T(\mathbf{s}, \mathbf{m}, \alpha) = T_{S\to G}^{-1} R_x(\alpha) T_{M\to G} \tag{8.2}$$

where $T_{S\to G} \in \mathrm{SE}(3)$ is a transformation with $T_{S\to G}(\mathbf{s}) = 0$ and $R(T_{S\to G})(\mathbf{n_s})) = (1,0,0)^T$, $T_{M\to G}$ is a corresponding transformation for $\mathbf{m}$ and $R_x(\alpha)$ is a rotation around the $x$-axis with angle $\alpha$. Note that $T_{S\to G}$ and $T_{M\to G}$ are not unique. We instead precompute one of the possible transformations for each scene and model point and keep it fixed.

For discretization, the rotation angle $\alpha$ is sampled in $d$ intervals, such that each vertex can be parametrized as $S \times (M \times \{0, 1, \ldots, d-1\})$. The number of vertices in the full graph is then $|S||M|d$.

**Discussion** Note that the degrees of freedom introduced by the deformations are not captured by this parametrization. We found that the proposed parametrization is only slightly slower but significantly more robust in 3D. Fig. 8.1 shows an example of how the extracted correspondences change when adding the angle to the parametrization.

### 8.2.3 Graph Creation and Local Voting Scheme

Handling a graph with $|S||M|d$ vertices can become computationally expensive for larger scenes. In order to improve the matching speed, we prune the graph based on the results of the local voting scheme of Sec. 5, thus effectively removing parts that we deem unlikely to be relevant. Fig. 8.2 outlines the graph creation.

Contrary to Sec. 5, we perform the voting for all reference points simultaneously. For each model point pair that matches a scene point pair, we obtain the symmetric parameter $\alpha_2$ and cast a vote for reference point $\mathbf{s}_2$ at $(\mathbf{m}_2, \alpha_2)$. The two corresponding nodes of the graph, $(\mathbf{s}_1, \mathbf{m}_1, \alpha_1)$ and $(\mathbf{s}_2, \mathbf{m}_2, \alpha_2)$, are connected with an edge, since they can both be fulfilled simultaneously. We create a sparse graph by adding only those vertices that have a high voting score. This removes vertices and edges that are unlikely to be a part of the object. In practice, for each scene reference point, we use the references with the highest 3% of voting scores.

Note that since the threshold is applied after the voting and before the actual graph creation, the removed vertices and edges are never actually created in memory. Instead, the graph is created as follows:

1. Perform the voting scheme for each scene point $\mathbf{s} \in S$ as reference point and

find the peaks in the accumulator array (see Sec. 5.3.6). For each peak entry $(\mathbf{m}, \alpha)$, create the graph vertex $v = (\mathbf{s}, \mathbf{m}, \alpha)$ and add it to $V$.

2. Create the graph's edges by checking, for each pair of vertices $(v_1, v_2)$, if any of the trained deformations of the model point pair $(m(v_1), m(v_2))$ is equal to $(s(v_1), s(v_2))$, and if the corresponding rotation angles match. This effectively checks if a deforming transformation exists that fulfills the correspondences of both vertices.

The left images in Fig. 8.7 show an example of the pruned graph creation. For a full graph, each model vertex would be connected to each scene point. For our pruned graph, only a small subset of those connections remains. As outlined in Fig. 8.1, the pruning step improves the runtime of the graph matching by several orders of magnitude.

Note that our approach is an effective alternative to using feature point descriptors to find potentially matching scene and model points: Instead of computing such features for both point clouds and comparing them, we use the voting to detect potentially similar points.

### 8.2.4 Graph Matching

In the following, we follow the notation of [48]. The challenge is to find an assignment vector $\hat{X} \in \{0,1\}^V$, where $\hat{X}_v = 1$ if the scene and model point represented by $v$ correspond and $\hat{X}_v = 0$ otherwise. This problem is relaxed, *i.e.*, we allow continuous values for the assignments, and write $X \in (R^+)^V$ for the relaxed assignment vector.

The relaxed assignment is modeled as an energy optimization problem

$$X^* = \arg\max_{|X|=1} \sum_{e=(v_i,v_j)\in E} X_{v_i} X_{v_j} \tag{8.3}$$

Intuitively, an active vertex only contributes positively to the total energy if it is connected to another active vertex. Since the total amount of activations is limited through $|X| = 1$, the energy is maximized if the activated vertices are strongly connected.

In terms of the graph's adjacency matrix $A = (w_{i,j})$, (8.3) becomes

$$X^* = \arg\max_{|X|=1} \sum_{i,j\in V} w_{i,j} X_{v_i} X_{v_j} \tag{8.4}$$

$$= \arg\max_{|X|=1} X^T A X \tag{8.5}$$

Note that for the normalization $|X| = 1$, any norm can be used, since we will use the relative values of $X$ only. The problem is then a scaled Rayleight quotient problem [82, 48], and $X^*$ is an eigenvector associated to the largest eigenvalue of $A$.

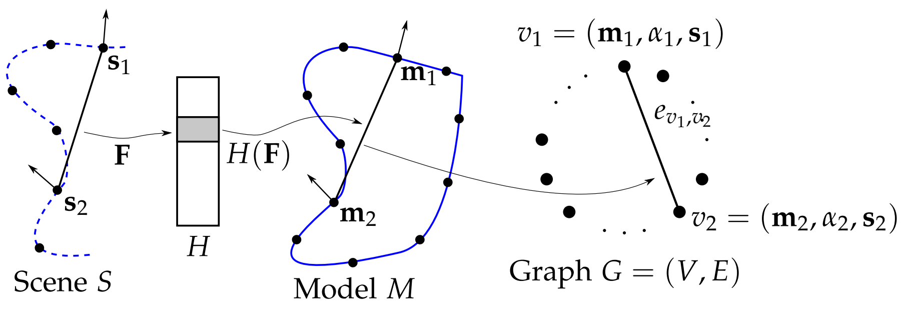


Figure 8.2: **Graph Construction**. *From left to right*: For each scene point pair $(\mathbf{s}_1, \mathbf{s}_2)$, $\mathbf{F}$ is computed. The hash table returns a list $H(\mathbf{F})$ of all model point pairs that can be deformed to match $(\mathbf{s}_1, \mathbf{s}_2)$. *Right*: Each vertex $v$ in the graph represents a possible correspondence between a scene and a model point. Edges are created between vertices that are consistent, *i.e.*, a deformable transformation between scene and model exists that fulfills both correspondences: For each match $(\mathbf{m}_1, \mathbf{m}_2) \in H(\mathbf{F})$, an edge is created.

We solve the optimization problem through gradient descend. $X^0$ is initialized to all ones, the update step is

$$X^{k+1} = \frac{AX^k}{|AX^k|} \tag{8.6}$$

This is equivalent to the power iteration that has proven convergence against an eigenvector of the largest eigenvalue of $A$.

**Voting Scheme Interpretation** The iteration (8.6) can also be seen as a repeated, re-weighted voting scheme: In the first step, each vertex votes for all connected vertices with a weight of 1, such that $X^1_v$ is the degree of $v$, *i.e.*, the number of connected edges. In subsequent steps, each vertex $v$ votes again for all connected vertices, but this time with the number of votes it received in the last round, instead of 1. Through this feedback cycle, vertices of a strongly connected subgraph amplify each other, while the values of weakly connected vertices decrease due to normalization. With this interpretation, the graph pruning is equivalent to performing the first iteration of (8.6) on the full graph and then removing vertices with low scores.

### 8.2.5 Dominant Consistent Subgraph Extraction

The power iteration returns a vector of weights $X^*$ for vertices or scene-model correspondences. However, even though the correct correspondences obtain high weights, there is no guarantee that all incorrect correspondences have very low weights. When performing a simple threshold on $X^*$, outliers as well as non-unique correspondences, *i.e.*, two or more connections to a model or scene point, might be included. If more than one instance of the object with similar size is present in the scene, their correspondences might be mixed together, having

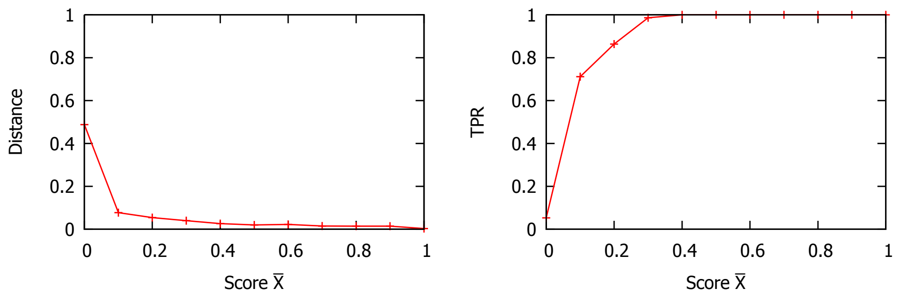


Figure 8.3: Correlation between the weight $\overline{X}$ and the correctness of a correspondence. *Left:* Average distance of a corresponding scene point from the ground truth scene point, relative to the diameter of the model. *Right:* Ratio of correct correspondences w.r.t. the score $\overline{X}$.

correspondences of both instances with high scores. An additional step must thus be performed to ensure that the correspondences are consistent.

In [82], a greedy approach for extracting the most dominant, consistent dense subgraph was proposed. For this, starting with the strongest correspondence $v^*$, the strongest remaining correspondence that does not contradict any previously added correspondences is added iteratively. However, this approach is computationally expensive since many consistency checks must be performed. Additionally, it requires a strong deformation model, which is not available to us. [48] modeled the optimization based on the $l_1$-norm, giving an almost binary correspondence vector, which is easier to threshold. However, we found that this approach has a slower convergence and tends to drop correct nodes.

We instead use a simple greedy subgraph extraction. Similar to [82], we use the strongest correspondence $v^* = \arg\max_{v \in V} X^*(v)$ as anchor, assuming that it is correct. All other vertices $v$ are weighted based on their distance to $v^*$, and on the weights of the vertices on the shortest path between $v$ and $v^*$. Formally, let

$$P(v_a, v_b) = \{(v_a, v_1, v_2, \dots, v_i, v_b) : (v_a, v_1), (v_1, v_2), \dots (v_i, v_b) \in E\} \tag{8.7}$$

be the set of all paths between $v_a$ and $v_b$. The weight $\overline{X}_v$ of $v$ is then

$$\overline{X}_v = \max_{p \in P(v, v^*)} \prod_{v' \in p} \alpha X_{v'} \tag{8.8}$$

where $0 < \alpha \leq 1$ is a damping factor that additionally downweights vertices further away from $v^*$. The intuition behind $\alpha$ is that even though the subgraph of interest is not fully connected, it is dense such that its vertices are connected with short paths. Vertices far away from $v^*$ are thus less likely to be part of the desired subgraph. In all our experiments, we set $\alpha = 0.5$.

To avoid double-correspondences of scene or model points, if a scene of model point is part of two or more extracted correspondences, we only keep the correspondence with the highest value in $\overline{X}$. We found that such double-correspondences mostly connect two neighboring points of one set to a single point in the other set, a result of the allowed deformation. In practice, we limit the length of the paths to 4. This aids performance: starting with $v^*$, only a small part of the graph must be explored. We also threshold the correspondences, keeping only those with $\overline{X}_v > \tau_X = 0.05$.

The result of this method is a new set of vertex weights $\overline{X}$ that can be seen as a consistent soft assignment between model and scene points. It thus keeps the idea of soft, weighted assignments over hard, binary ones, but is more likely to be consistent. $\overline{X}$ can then be used to initialize a deformable refinement.

Fig. 8.3 further motivates this approach. It shows the correlation between the score $\overline{X}_v$ and the correctness of the correspondence represented by $v$ for an example scene, using $\alpha = 1$. Note that for scores larger than $\approx 0.3$, the correspondences are almost exclusively correct. The method successfully recomputes votes to recover consistent subgraph that follows the most dominant correspondence.

## 8.3 Experiments

We evaluated the proposed approach with several quantitative and qualitative experiments. Synthetic and real data with available ground truth was used for the quantitative evaluation, while the qualitative experiments were performed on a real dataset only.

Note that all parameters were kept constant over all experiments, showing the method's robustness w.r.t. its parameters. Model and scene were subsampled with distance 3% of the model's diameter. For the hash table, the distance of feature **F** was quantized in steps of 5% of the model's diameter, while angles were quantized in steps of 12°. Fig. 8.5 motivates the choice for the distance sampling parameter, which is a tradeoff between matching accuracy and matching speed. For each scene, 10 iterations of (8.6) were performed.

The method was implemented in C, timings were measured on an Intel Core i5 with 3.33 GHz and 16 GB RAM. The offline learning phase, *i.e.*, creation of the hash table $H$, took less than 1 minute for all objects. Feature matching required 0.05 to 2 seconds, the power iterations 0.1 to 2.5 seconds, depending on the complexity of the scene and the amount of clutter. Timings for the remaining steps, such as scene sampling and greedy dense subgraph extraction, were negligible. We believe that an improved implementation and a better control over the number of iterations would significantly improve the runtime.

### 8.3.1 Quantitative

**Synthetic data** A first set of experiments was performed on synthetic data, where ground truth is available. We selected three different objects with different surface characteristics: a clamp and a pipe joint from MVTec and the Stanford Bunny [124]

Table 8.1: Effect of matching with a sparse graph using the local voting scheme for the scene shown in Fig. 8.7

| | $\lvert S\rvert$ | $\lvert M\rvert$ | Vertices $\lvert V\rvert$ | Edges $\lvert E\rvert$ | Runtime |
|---|---|---|---|---|---|
| Dense | 13106 | 300 | 135.566 | 98.886.050 | 1163.6 s |
| Sparse | 13106 | 300 | 34.095 | 42.832 | 1.1 s |

Table 8.2: Average precision, recall, and relative error of the returned correspondences for the synthetic scenes

| Model | Precision | Recall | Rel. Error |
|---|---|---|---|
| Clamp | 0.93 | 0.57 | 3.6% |
| Pipe joint | 0.99 | 0.69 | 2.2% |
| Bunny | 0.96 | 0.51 | 4.1% |

(Fig. 8.4, top). For each object, 100 scenes were rendered with different amounts of clutter, occlusion, and deformation (Fig. 8.4, bottom). The objects were deformed using free-form deformation [125]. For training, 10 deformed instances of each object, which were not part of any of the evaluation scenes, were used.

We measure the performance of the method in terms of precision, recall, and error of the recovered correspondences. A recovered correspondence is a true positive if its scene point is on the object and its model point is at most 10% away from its ground truth position. The relative error measures for each true positive correspondence the distance of the corresponding model point to the ground truth model point, divided by the diameter of the object.

Tab. 8.2 shows the average results for the three objects. The recovered correspondences show a very high precision, indicating that most of the recovered correspondences were correct. The average recall is larger than 0.5, meaning that on average more than half of the correct correspondences were recovered. We believe that those are enough correspondences to initialize a deformable ICP.

**Real data** We further evaluated our approach on the dataset of Mian *et al*. [92, 93] (compare Sec. 5.4.2). Since the objects of that dataset are rigid, we trained with a single model only. Fig. 8.6 shows the detection rates w.r.t. the occlusion of the objects. See also Fig. 5.23c.

Note that even though the objects are rigid, detection still benefits from using our graph approach, as is evident from the fact that we exceed the baseline method of Sec. 5, which we use to initialize our graph. This shows that multiple iterations of the voting, as discussed in Sec. 8.2.4, lead to a more robust result. We also outperform several other state of the art methods. Note that the dataset is in practice maxed out, as only 7 object instances remain undetected. Those instances are almost completely occluded and difficult to detect even by humans.

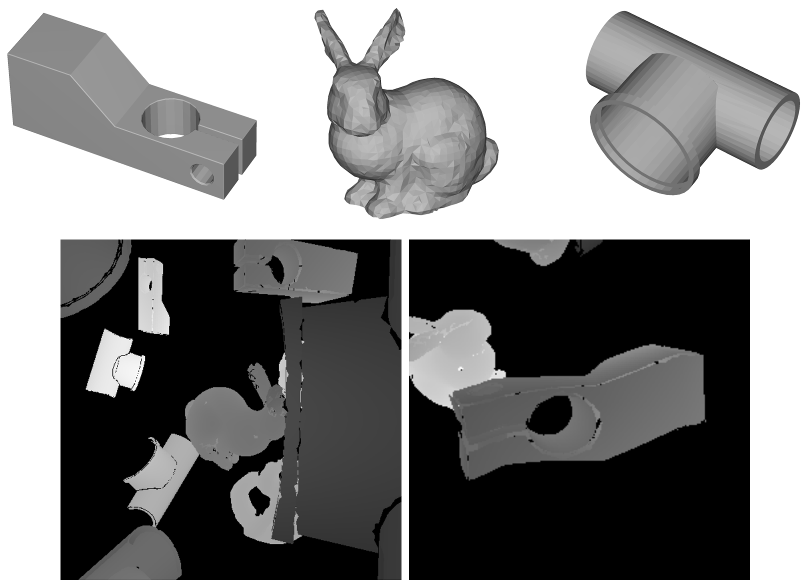


Figure 8.4: **Top**: Objects used for the synthetic tests (clamp, bunny, pipe joint). **Bottom**: Example scenes of the synthetic dataset, showing clutter and deformation.

### 8.3.2 Qualitative

We evaluated the proposed method on a set of real-world scenarios. Over 50 scenes containing pretzels, bananas, caps, stressballs and a silicone baking mold were acquired using both an industrial stereo sensor and a Primesense RGB-D sensor and matched against the corresponding model. Note that since the stereo sensor does not return an RGB-image, its scenes are visualized in 3D only.

For training, several deformed instances of each object were acquired, manually segmented and registered using deformable ICP [101]. We used only 5 to 15 examples for each class for the training, showing that the method is able to generalize from only few examples.

Fig. 8.8 and 8.9 show several example scenes. Tab. 8.7 shows on two examples how the graph creation leads to a sparse graph and how the graph matching extracts a consistent set of correspondences from that graph. The effect on the computational costs are shown in Tab. 8.1.

Overall, we found that the method performs very well even in cases of severe clutter, occlusion, and noise.

## 8.4 Conclusion

This chapter introduced a deformable 3D object detection scheme that generalizes well over different object classes and requires only few parameters. The graph matching scheme of [82] was extended by augmenting the correspondences with another parameter, making them more expressive in 3D. We prune the graph by

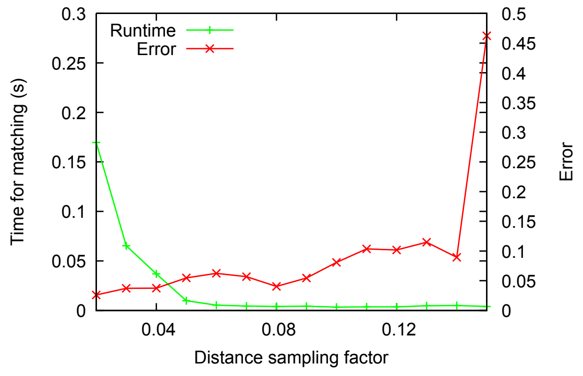


Figure 8.5: Effect of changing the distance sampling parameter of the feature database for an exemplary synthetic scene. Matching accuracy and robustness drops significantly when sampling with more than 0.1, while matching time raises significantly when sampling with less than 0.05. In practice, we use 0.05 over all our experiments.

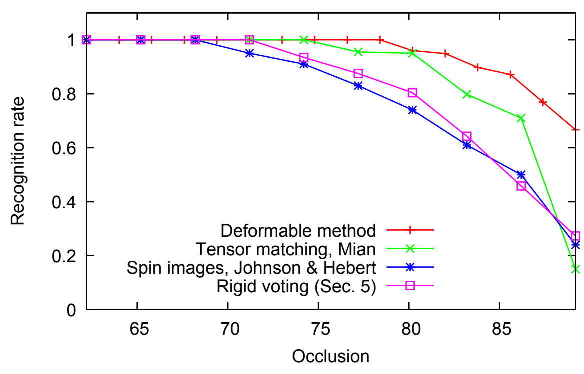


Figure 8.6: Detection results on the dataset of Mian *et al.* [91]. Our approach exceeds the rigid baseline method of Sec. 5 and successfully detects 96.3% (181 of 188) of all objects, and 98.8% (168 of 170) of objects with less than 84% occlusion. The deformable method also outperforms spin images of Johnson and Hebert [75] and the tensor voting of Mian *et al.* [92].

using the method of [46] to create only a sparse set of correspondences that are likely to be correct. Using 3D point pairs makes the method invariant against any rigid 3D transformations. Finally, a greedy dense subgraph extraction is proposed to recover a consistent set of correspondences, which can be used to obtain an approximate rigid transformation or to initialize a deformable ICP.

The method can also be seen as an extension of the voting scheme of Sec. 5 into an iterative, re-weighted voting scheme.

Our experiments show that the proposed method is able to robustly detect rigid and non-rigid objects in challenging 3D point clouds despite heavy clutter and partial object occlusion. They also showed how the combination of all possible deformations can be learned based on only a few deformed training samples. For rigid objects, we outperform prior art.

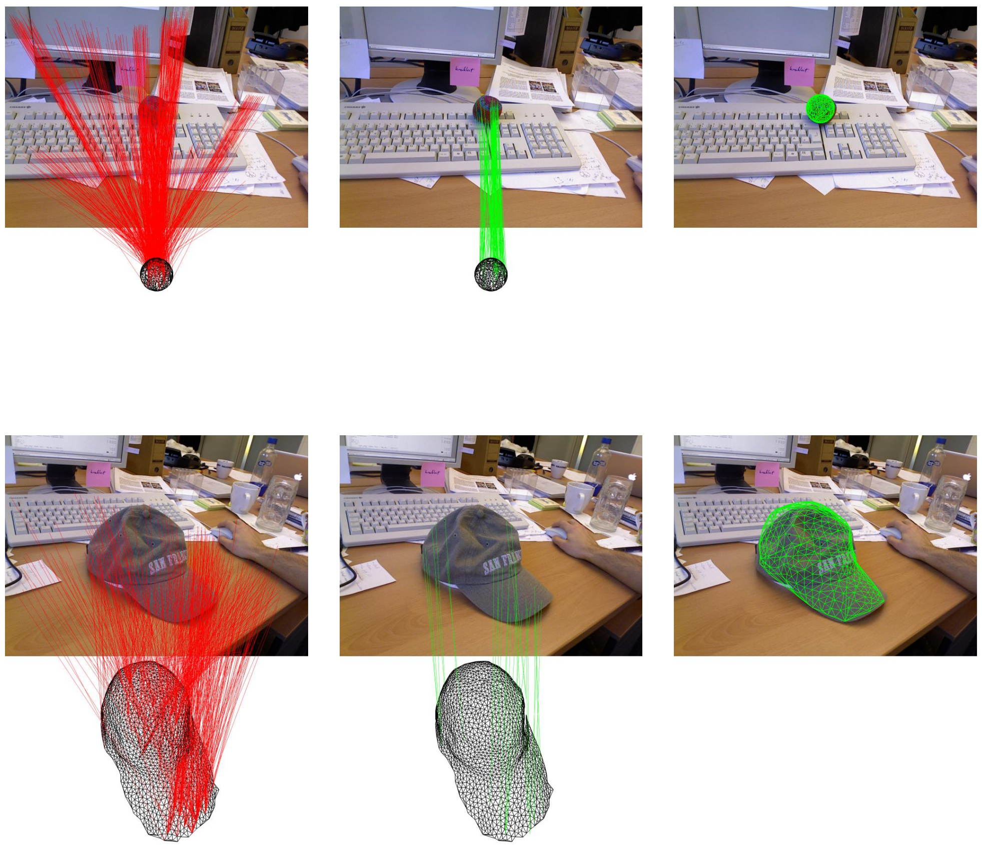


Figure 8.7: Graph matching examples. **Left**: Initial correspondences, created by thresholding the results of the local voting scheme. Each correspondence is a vertex in our graph. **Center**: Correspondences extracted after graph matching by the greedy subgraph extraction. Note that only a consistent set of correspondences from the original set of correspondences remains. **Right**: The correspondences were transformed into a rigid transformation. The matching was performed on the depth image only, while the RGB image is used for visualization only.

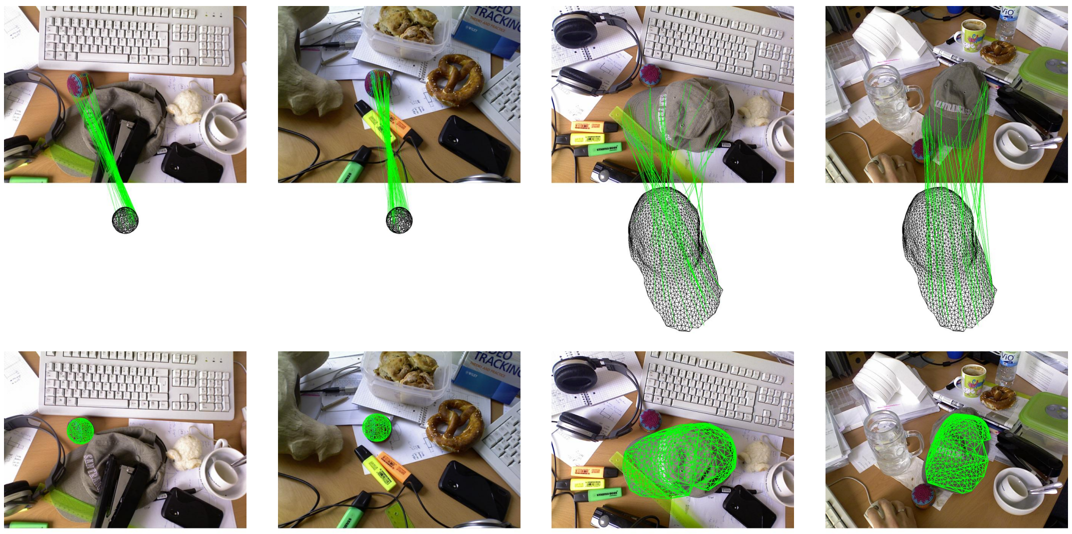


Figure 8.8: Qualitative graph matching results on scenes acquired with a Primesense sensor. **Top**: Scenes and recovered correspondences. **Bottom**: Rigid transformation of the base model. The transformation was recovered from the correspondences. Note that for stronger deformations that were not trained, still enough correspondences were found for an approximate matching (rightmost scene).

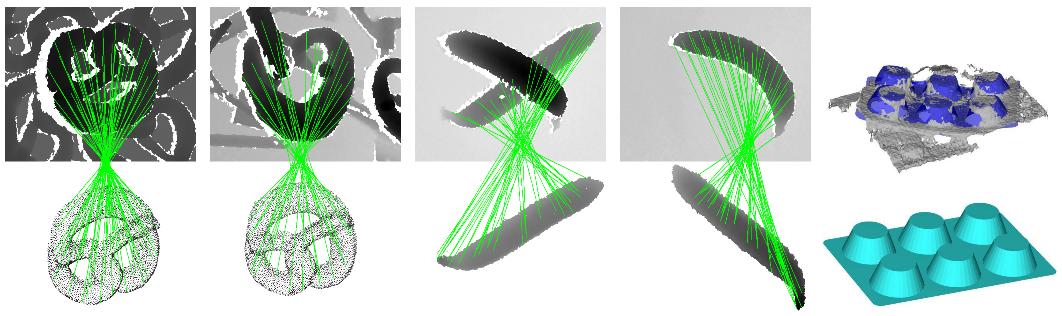


Figure 8.9: Qualitative results on scenes acquired with a stereo sensor. Challenges include clutter, occlusion, multiple instances, and strong deformations. The rightmost scene shows the model (bottom) and fitted result (top).

# 9 Conclusion

## 9.1 Summary

This thesis tackled the challenge of developing 3D machine vision algorithms that allow solving a large variety of real-world problems arising in industrial and robotic settings. The algorithms were developed with the requirements of a generic industrial machine vision library in mind and are generic, easy to use and to parametrize, work with data from different sensors and with different characteristics, are fast, robust, and accurate.

The main contributions of this thesis are

- a novel, high-performance 3D nearest neighbor algorithm that allows lookups in almost constant time ($\mathcal{O}(\log\log N)$ for a target cloud of $N$ points). The lookup times are in particular almost independent of scene and query point distributions. The method outperforms state-of-the-art methods such as the $k$-d-tree and the ANN and FLANN libraries;
- an industrial-grade implementation of ICP that is generic, robust, fast, accurate and easy to use;
- a novel rigid 3D object detection framework that employs a voting scheme on a local, data-driven parameter space using point pairs as features. The voting is wrapped in a detection pipeline that removes duplicates and further refines, verifies and rates the results. The framework allows detecting objects of any shape and is fast, accurate, and robust against even large amounts of clutter, noise, and occlusion. Detection times of 190 ms for 10 object instances, including refinement, enable manipulation tasks such as bin-picking as well as inspection tasks such as surface comparisons;
- an adaption of the baseline voting scheme to multimodal data, such as RGB-D images, that simultanously matches the 3D surface and the 2D sillhouette of the object by combining both into a novel, multimodal feature;

- an adaption of the baseline voting scheme to primitive 3D shapes – planes, spheres, and cylinders – that uses the intrinsic symmetries of those shapes to further reduce the parameter space and to improve the feature matching. Additionally, an adaption of the ICP method robustly refines these primitives in the scene point cloud;
- an adaption of the baseline voting scheme to deformable objects that uses the voting and the local parametrization to enable a robust and efficient graph matching in 3D.

When combined, these methods allow solving a large variety of real-world problems such as bin picking, 3D surface inspection and defect detection, robot navigation, and others.

Several of the methods developed in this thesis were implemented in the machine vision library HALCON [100] and are used in industrial and robotic installations worldwide.

## 9.2 Dependent Work

At the time of finishing this thesis, several other methods were published that use or extend the object detection algorithms developed in this work.

**Applications** Salas-Moreno *et al.* [118] introduced the simultaneous localization and mapping algorithm *SLAM++*, which, among other contributions, parallelizes the voting proposed in Sec. 5 on a GPU. They collect the votes in lists by augmenting the accumulator space index with the scene point index. The lists are sorted, and the number of repetitions of a particular entry equals its number of votes. Chliveros *et al.* [26] use the rigid detector to initialize a tracking process. Beetz *et al.* [11] use the rigid detector to locate the cooking tools held by a robot in a kitchen environment. Klank *et al.* [79] embed the rigid detection method in a robotic framework that automatically evaluates different detection methods, allowing the robot to select the best performing method based on object, sensors and context.

**Pipeline Modifications** Similar to Sec. 7, de Figueiredo *et al.* [37, 38] extend the rigid detection method to solids of revolution by exploiting the symmetries of such shapes to reduce the parameter space and to improve the feature training and matching.

Skotheim *et al.* [127] extended the rigid detection pipeline twofold. For preprocessing, outlier points are removed from the scene before the matching. For post-processing, the detected poses are verified using given constraints that limit the set of possible poses. For example, certain objects would always be expected to stand upright or to be on a table. The poses are further verified by rendering them in an artificial depth image and comparing that rendering to the depth image that represents the scene.

Tutzel *et al*. [147] proposed to weight the different point pairs depending on their significance for the result. For example, for almost symmetric objects, point pairs where one of the points lies on the object part that breaks the symmetry would vote with a higher weight, which helps to find the correct among similar poses. The weights are obtained such that the detection is optimized over several scenes with known ground truth.

Birdal and Ilic [17] also proposed a re-weighting of the point pairs. However, they do so in a scene-independent way, giving stronger weights to points in more planar areas whose normals are more robust and repeatable to compute. This aids both accuracy and detection performance. Additionally, they pre-segment the scene such that objects are unlikely to be part of two segments simultaneously, allowing again for a faster and more robust voting. Finally, they add a more sophisticated hypothesis verification that takes visibility constraints into account.

**Point Pair Feature Extensions** Several other approaches modify the point pair features in various ways. Kim and Medioni [77] proposed to extend the 3D point pair feature with a *visibility context* that indicates if a point pair can be visible in a certain configuration. Additionally, the add information about the visible boundary of the object to the point pairs, effectively building a viewpoint dependent point pair that matches both surface and contour. Choi *et al*. [29] proposed to detect and use edges and line segments in range images, and to form point pair features that include those edges or line segments. This is similar to the multimodal features described in Sec. 6. It is, however, limited to range sensors that provide very accurate information at edges and to objects that have articulated edges.

Choi and Christensen [27] modify the method of Sec. 5 by extending the 3D point pair feature with information about the color of the two points. They later parallelize the method on the GPU [28] and perform an extensive evaluation, showing improved performance on multimodal data when detecting colored every-day objects. Similar, McElhone *et al*. [89] augment the 3D point pair feature with local color information and extend the rigid object detector to multiple resolutions. Nguyen *et al*. [102] use a different variant of the point pair feature of Sec. 5.3.1 that resolves the mirroring ambiguity. They also compare normal directions between scene and aligned model to evaluate and verify matches.

## 9.3 Future Work

While the proposed methods are important building blocks for solving many industrial applications, many more remain.

The multimodal detection method of Sec. 6 would benefit from a refinement method that simultaneously refines both the 3D surface overlap and the alignment of 2D edges. This would help to properly align planar objects that can otherwise slide when using point-to-plane metrics. Similar, a parameter and model free deformable refinement that allows sparse initialization would help to further improve the results of the deformable detections of Sec. 8.

Another often required method is the detection and fitting of 3D boxes in 3D or multimodal data. Such methods can be used for several applications in warehousing and logistics, such as automatic unloading of containers and trucks, stacking and packing of boxes for further transportation or storage, or quality inspection of boxes. The main challenges are the larger number of shape parameters (the three dimensions of the box), possible deformations such as bent sides, problematic viewpoints where only one or two sides are visible, and the fact that for a proper determination of the dimensions, edges must be taken into account unless multiple sensors are used.

Another challenge is the simultaneous classification and detection of multiple, potentially deformable or non-rigid, different objects in a scene. Applications include warehousing, where a robot sees multiple objects at once which need to be identified, robotics in less structured environments, where scene understanding is helpful for operational planning, and retail and logistics, where unordered piles of different objects need to be classified or sorted.

# Bibliography

[1] Sung Joon Ahn, Ira Effenberger, Sabine Roth-Koch, and Engelbert Westkämper, *Geometric Segmentation and Object Recognition in Unordered and Incomplete Point Cloud*, in Pattern Recognition, 25th DAGM Symposium, Magdeburg, Germany, September 10-12, 2003, Proceedings, Bernd Michaelis and Gerald Krell, eds., vol. 2781 of Lecture Notes in Computer Science, Springer, 2003, pp. 450–457.

[2] Aitor Aldoma, Federico Tombari, Luigi di Stefano, and Markus Vincze, *A Global Hypotheses Verification Method for 3D Object Recognition*, in Computer Vision - ECCV 2012 - 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part III, Andrew W. Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, eds., vol. 7574 of Lecture Notes in Computer Science, Springer, 2012, pp. 511–524.

[3] Dragomir Anguelov, Praveen Srinivasan, Hoi-Cheung Pang, Daphne Koller, Sebastian Thrun, and James Davis, *The Correlated Correspondence Algorithm for Unsupervised Registration of Nonrigid Surfaces*, in Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada], 2004, pp. 33–40.

[4] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu, *An Optimal Algorithm for Approximate Nearest Neighbor Searching Fixed Dimensions*, Journal of the ACM (JACM), 45 (1998), pp. 891–923.

[5] Marco Attene, Bianca Falcidieno, and Michela Spagnuolo, *Hierarchical mesh segmentation based on fitting primitives*, The Visual Computer, 22 (2006), pp. 181–193.

[6] W. W. Rouse Ball, *Mathematical Recreations and Essays*, Macmillan, New York, 1892.

[7] Dana H. Ballard, *Generalizing the Hough transform to detect arbitrary shapes*, Pattern Recognition, 13 (1981), pp. 111–122.

[8] Prabin Bariya and Ko Nishino, *Scale-hierarchical 3D object recognition in cluttered scenes*, in The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010, IEEE Computer Society, 2010, pp. 1657–1664.

[9] Norbert. Bauer, Norbert. Albrecht, and Fraunhofer-Allianz Vision., *Guideline industrial image processing*, Fraunhofer-Allianz Vision, Stuttgart, 2003.

[10] Albert E Beaton and John W Tukey, *The fitting of power series, meaning polynomials, illustrated on band-spectroscopic data*, Technometrics, 16 (1974), pp. 147–185.

[11] Michael Beetz, Ulrich Klank, Ingo Kresse, Alexis Maldonado, Lorenz Mösenlechner, Dejan Pangercic, Thomas Rühr, and Moritz Tenorth, *Robotic roommates making pancakes*, in 11th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2011), Bled, Slovenia, October 26-28, 2011, IEEE, 2011, pp. 529–536.

[12] Richard Ernest Bellman, *Adaptive control processes: a guided tour*, vol. 4, Princeton university press Princeton, 1961.

[13] Jon Louis Bentley, *Multidimensional Binary Search Trees Used for Associative Searching*, Communications of the ACM (CACM), 18 (1975), pp. 509–517.

[14] Alexander C. Berg, Tamara L. Berg, and Jitendra Malik, *Shape Matching and Object Recognition Using Low Distortion Correspondences*, in 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA, IEEE Computer Society, 2005, pp. 26–33.

[15] Paul J. Besl and Ramesh Jain, *Three-Dimensional Object Recognition*, ACM Computing Surveys (CSUR), 17 (1985), pp. 75–145.

[16] Paul J. Besl and Neil D. McKay, *A Method for Registration of 3-D Shapes*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 14 (1992), pp. 239–256.

[17] Tolga Birdal and Slobodan Ilic, *Point Pair Features Based Object Detection and Pose Estimation Revisited*, in 2015 International Conference on 3D Vision, 3DV 2015, Lyon, France, October 19-22, 2015, Michael S. Brown, Jana Kosecká, and Christian Theobalt, eds., IEEE, 2015, pp. 527–535.

[18] Marcel Birn, Manuel Holtgrewe, Peter Sanders, and Johannes Singler, *Simple and Fast Nearest Neighbor Search*, in Proceedings of the Twelfth Workshop on Algorithm Engineering and Experiments, ALENEX 2010, Austin, Texas, USA, January 16, 2010, Guy E. Blelloch and Dan Halperin, eds., SIAM, 2010, pp. 43–54.

[19] Imma Boada, Narcís Coll, Narcis Madern, and Joan Antoni Sellarès, *Approximations of 3D generalized Voronoi diagrams*, in (Informal) Proceedings of the 21st European Workshop on Computational Geometry, Eindhoven, The Netherlands, March 9-11, 2005, Technische Universiteit Eindhoven, 2005, pp. 163–166.

[20] Imma Boada, Narcís Coll, Narcis Madern, and Joan Antoni Sellarès, *Approximations of 2D and 3D generalized Voronoi diagrams*, International Journal of Computer Mathematics, 85 (2008), pp. 1003–1022.

[21] Dorit Borrmann, Jan Elseberg, Kai Lingemann, and Andreas Nüchter, *The 3D Hough Transform for plane detection in point clouds: A review and a new accumulator design*, 3D Research, 2 (2011), pp. 1–13.

[22] Richard J. Campbell and Patrick J. Flynn, *A Survey Of Free-Form Object Representation and Recognition Techniques*, Computer Vision and Image Understanding, 81 (2001), pp. 166–210.

[23] John Canny, *A Computational Approach to Edge Detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 8 (1986), pp. 679–698.

[24] Hui Chen and Bir Bhanu, *3D free-form object recognition in range images using local surface patches*, Pattern Recognition Letters, 28 (2007), pp. 1252–1262.

[25] Michael Chertok and Yosi Keller, *Efficient High Order Matching*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 32 (2010), pp. 2205–2215.

[26] Georgios Chliveros, Rui Pimentel de Figueiredo, Plinio Moreno, Maria Pateraki, Alexandre Bernardino, José Santos-Victor, and Panos E. Trahanias, *A Framework for 3D Object Identification and Tracking*, in VISAPP 2014 - Proceedings of the 9th International Conference on Computer Vision Theory and Applications, Volume 3, Lisbon, Portugal, 5-8 January, 2014, Sebastiano Battiato and José Braz, eds., SciTePress, 2014, pp. 672–677.

[27] Changhyun Choi and Henrik I. Christensen, *3D pose estimation of daily objects using an RGB-D camera*, in 2012 IEEE/RSJ International Conference on

Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012, IEEE, 2012, pp. 3342–3349.

[28] Changhyun Choi and Henrik I. Christensen, *RGB-D object pose estimation in unstructured environments*, Robotics and Autonomous Systems, 75 (2016), pp. 595–613.

[29] Changhyun Choi, Yuichi Taguchi, Oncel Tuzel, Ming-Yu Liu, and Srikumar Ramalingam, *Voting-based pose estimation for robotic assembly using a 3D sensor*, in IEEE International Conference on Robotics and Automation, ICRA 2012, 14-18 May, 2012, St. Paul, Minnesota, USA, IEEE, 2012, pp. 1724–1731.

[30] Won-Seok Choi and Se-Young Oh, *Fast Nearest Neighbor Search using Approximate Cached k-d tree*, in 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012, IEEE, 2012, pp. 4524–4529.

[31] Chin-Seng Chua and Ray Jarvis, *Point Signatures: A New Representation for 3D Object Recognition*, International Journal of Computer Vision (IJCV), 25 (1997), pp. 63–85.

[32] Haili Chui and Anand Rangarajan, *A new point matching algorithm for non-rigid registration*, Computer Vision and Image Understanding, 89 (2003), pp. 114–141.

[33] Anna Clark, *The struggle for the breeches gender and the making of the British working class*, University of California Press, Berkeley, 1995.

[34] John G. Cleary and Geoff Wyvill, *Analysis of an algorithm for fast ray tracing using uniform space subdivision*, The Visual Computer, 4 (1988), pp. 65–83.

[35] Fernand S. Cohen and Jin-Yinn Wang, *Part II: 3-D Object Recognition and Shape Estimation from Image Contours Using B-splines, Shape Invariant Matching, and Neural Network*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 16 (1994), pp. 13–23.

[36] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento, *Thirty Years Of Graph Matching In Pattern Recognition*, International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI), 18 (2004), pp. 265–298.

[37] Rui Pimentel de Figueiredo, Plinio Moreno, and Alexandre Bernardino, *Fast 3D Object Recognition of Rotationally Symmetric Objects*, in Pattern Recognition and Image Analysis - 6th Iberian Conference, IbPRIA

2013, Funchal, Madeira, Portugal, June 5-7, 2013. Proceedings, João M. Sanches, Luisa Micó, and Jaime S. Cardoso, eds., vol. 7887 of Lecture Notes in Computer Science, Springer, 2013, pp. 125–132.

[38] Rui Pimentel de Figueiredo, Plinio Moreno, and Alexandre Bernardino, *Efficient pose estimation of rotationally symmetric objects*, Neurocomputing, 150 (2015), pp. 126–135.

[39] B Delaunay, *Sur la sphere vide. A la memoire de George Voronoi*, Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et na, (1934), pp. 793–800.

[40] Chitra Dorai and Anil K. Jain, *COSMOS - A Representation Scheme for 3D Free-Form Objects*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 19 (1997), pp. 1115–1130.

[41] Bertram Drost and Slobodan Ilic, *3D Object Detection and Localization Using Multimodal Point Pair Features*, in 2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission, Zurich, Switzerland, October 13-15, 2012, IEEE Computer Society, 2012, pp. 9–16.

[42] Bertram Drost and Slobodan Ilic, *A Hierarchical Voxel Hash for Fast 3D Nearest Neighbor Lookup*, in Pattern Recognition - 35th German Conference, GCPR 2013, Saarbrücken, Germany, September 3–6, 2013. Proceedings, Joachim Weickert, Matthias Hein, and Bernt Schiele, eds., vol. 8142 of Lecture Notes in Computer Science, Springer, 2013, pp. 302–312.

[43] Bertram Drost and Slobodan Ilic, *Graph-Based Deformable 3D Object Matching*, in Pattern Recognition - 37th German Conference, GCPR 2015, Aachen, Germany, October 7-10, 2015, Proceedings, Juergen Gall, Peter V. Gehler, and Bastian Leibe, eds., vol. 9358 of Lecture Notes in Computer Science, Springer, 2015, pp. 222–233.

[44] Bertram Drost and Slobodan Ilic, *Local Hough Transform for 3D Primitive Detection*, in 2015 International Conference on 3D Vision, 3DV 2015, Lyon, France, October 19-22, 2015, Michael S. Brown, Jana Kosecká, and Christian Theobalt, eds., IEEE, 2015, pp. 398–406.

[45] Bertram Drost and Markus Ulrich, *Recognition and pose determination of 3D objects in multimodal scenes*, 2012. EP 2720171.

[46] Bertram Drost, Markus Ulrich, Nassir Navab, and Slobodan Ilic, *Model globally, match locally: Efficient and robust 3D object recognition*, in The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010, IEEE Computer Society, 2010, pp. 998–1005.

[47] Bertram Heinrich Drost and Markus Ulrich, *Recognition And Pose Determination Of 3d Objects In 3d Scenes Using Geometric Point Pair Descriptors And The Generalized Hough Transform*, 2010. EP 2385483.

[48] Olivier Duchenne, Francis R. Bach, In-So Kweon, and Jean Ponce, *A Tensor-Based Algorithm for High-Order Graph Matching*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 33 (2011), pp. 2383–2395.

[49] Rex A. Dwyer, *Higher-Dimensional Voronoi Diagrams in Linear Expected Time*, Discrete & Computational Geometry, 6 (1991), pp. 343–367.

[50] Jan Elseberg, Stephane Magnenat, Roland Siegwart, and Andreas Nuechter, *Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration*, Journal of Software Engineering for Robotics, 3 (2012), pp. 2–12.

[51] Leonhard Euler, *FORMVLAE GENERALES PRO TRANSLATIONE QVACVNQVE CORPORVM RIGIDORVM*, Novi Commentarii Academiae Scientiarum Imperialis Petropolitanae, (1776), pp. 189–207.

[52] Martin A. Fischler and Robert C. Bolles, *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography*, Communications of the ACM, 24 (1981), pp. 381–395.

[53] Andrew W. Fitzgibbon, *Robust registration of 2D and 3D point sets*, Image and Vision Computing, 21 (2003), pp. 1145–1153.

[54] Dariu Gavrila and Vasanth Philomin, *Real-Time Object Detection for "Smart" Vehicles*, in ICCV, 1999, pp. 87–93.

[55] Natasha Gelfand, Niloy J. Mitra, Leonidas J. Guibas, and Helmut Pottmann, *Robust Global Registration*, in Third Eurographics Symposium on Geometry Processing, Vienna, Austria, July 4-6, 2005, Mathieu Desbrun and Helmut Pottmann, eds., vol. 255 of ACM International Conference Proceeding Series, Eurographics Association, 2005, pp. 197–206.

[56] Josiah Willard Gibbs, *Elements of Vector Analysis*, Tuttle, Morehouse & Taylor, New Haven, 1884.

[57] Andrew S. Glassner, *Space subdivision for fast ray tracing*, Computer Graphics and Applications, IEEE, 4 (1984), pp. 15–24.

[58] Paulo F. U. Gotardo, Olga Regina Pereira Bellon, and Luciano Silva, *Range Image Segmentation by Surface Extraction Using an Improved Robust Estimator*, in 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2003), 16-22 June 2003, Madison, WI, USA, IEEE Computer Society, 2003, pp. 33–38.

[59] Claus Gramkow, *On Averaging Rotations*, Journal of Mathematical Imaging and Vision, 15 (2001), pp. 7–16.

[60] Michael A. Greenspan and Guy Godin, *A Nearest Neighbor Method for Efficient ICP*, in 3rd International Conference on 3D Digital Imaging and Modeling (3DIM 2001), 28 May - 1 June 2001, Quebec City, Canada, IEEE Computer Society, 2001, pp. 161–170.

[61] Michael A. Greenspan and Mike Yurick, *Approximate K-D Tree Search for Efficient ICP*, in 4th International Conference on 3D Digital Imaging and Modeling (3DIM 2003), 6-10 October 2003, Banff, Canada, IEEE Computer Society, 2003, pp. 442–448.

[62] Yulan Guo, Mohammed Bennamoun, Ferdous Ahmed Sohel, Min Lu, and Jianwei Wan, *3D Object Recognition in Cluttered Scenes with Local Surface Features: A Survey*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 36 (2014), pp. 2270–2287.

[63] Sariel Har-Peled, *A Replacement for Voronoi Diagrams of Near Linear Size*, in 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA, IEEE Computer Society, 2001, pp. 94–103.

[64] Günter Hetzel, Bastian Leibe, Paul Levi, and Bernt Schiele, *3D Object Recognition from Range Images using Local Feature Histograms*, in 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001), with CD-ROM, 8-14 December 2001, Kauai, HI, USA, IEEE Computer Society, 2001, pp. 394–399.

[65] Stefan Hinterstoisser, Stefan Holzer, Cedric Cagniart, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit, *Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes*, in IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011, Dimitris N. Metaxas, Long Quan, Alberto Sanfeliu, and Luc J. Van Gool, eds., IEEE, 2011, pp. 858–865.

[66] Paul W Holland and Roy E Welsch, *Robust regression using iteratively reweighted least-squares*, Communications in Statistics-Theory and Methods, 6 (1977), pp. 813–827.

[67] Stefan Holzer, Stefan Hinterstoisser, Slobodan Ilic, and Nassir Navab, *Distance transform templates for object detection and pose estimation*, in 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA, IEEE Computer Society, 2009, pp. 1177–1184.

[68] Adam Hoover, Gillian Jean-Baptiste, Xiaoyi Jiang, Patrick J. Flynn, Horst Bunke, Dmitry B. Goldgof, Kevin W. Bowyer, David W. Eggert, Andrew W. Fitzgibbon, and Robert B. Fisher, *An Experimental Comparison of Range Image Segmentation Algorithms*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 18 (1996), pp. 673–689.

[69] Heinz Hopf, *Vektorfelder in n-dimensionalen Mannigfaltigkeiten*, Mathematische Annalen, 96 (1927), pp. 225–249.

[70] Heinz Hopf, *Systeme symmetrischer Bilinearformen und euklidische Modelle der projektiven Räume*, in Selecta Heinz Hopf, Springer, 1964, pp. 107–118.

[71] Du Q. Huynh, *Metrics for 3D Rotations: Comparison and Analysis*, Journal of Mathematical Imaging and Vision, 35 (2009), pp. 155–164.

[72] Yoonho Hwang, Bohyung Han, and Hee-Kap Ahn, *A fast nearest neighbor search algorithm by nonlinear embedding*, in 2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012, IEEE Computer Society, 2012, pp. 3053–3060.

[73] Yani Ioannou, Babak Taati, Robin Harrap, and Michael A. Greenspan, *Difference of Normals as a Multi-scale Operator in Unorganized Point Clouds*, in 2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission, Zurich, Switzerland, October 13-15, 2012, IEEE Computer Society, 2012, pp. 501–508.

[74] Xiaoyi Jiang and Horst Bunke, *Edge Detection in Range Images Based on Scan Line Approximation*, Computer Vision and Image Understanding, 73 (1999), pp. 183–199.

[75] Andrew Edie Johnson and Martial Hebert, *Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 21 (1999), pp. 433–449.

[76] Dimitrios Katsoulas, *Robust Extraction of Vertices in Range Images by Constraining the Hough Transform*, in Pattern Recognition and Image Analysis, First Iberian Conference, IbPRIA 2003, Puerto de Andratx, Mallorca, Spain, June 4-6, 2003, Proceedings, Francisco J. Perales López, Aurélio C. Campilho, Nicolas Pérez de la Blanca, and Alberto Sanfeliu, eds., vol. 2652 of Lecture Notes in Computer Science, Springer, 2003, pp. 360–369.

[77] Eunyoung Kim and Gérard G. Medioni, *3D object recognition in range images using visibility context*, in 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2011, San Francisco, CA, USA, September 25-30, 2011, IEEE, 2011, pp. 3800–3807.

[78] Sung Il Kim and Sung Joon Ahn, *Extraction of Geometric Primitives from Point Cloud Data*, Proceedings of the International Conference on Control, Automation and Systems (ICCAS), 2005, 2 (2005), pp. 2010–2014.

[79] Ulrich Klank, Lorenz Mösenlechner, Alexis Maldonado, and Michael Beetz, *Robots that validate learned perceptual models*, in IEEE International Conference on Robotics and Automation, ICRA 2012, 14-18 May, 2012, St. Paul, Minnesota, USA, IEEE, 2012, pp. 4456–4462.

[80] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox, *Sparse distance learning for object recognition combining RGB and depth information*, in IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011, IEEE, 2011, pp. 4007–4013.

[81] Jungmin Lee, Minsu Cho, and Kyoung Mu Lee, *Hyper-graph matching via reweighted random walks*, in The 24th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2011, Colorado Springs, CO, USA, 20-25 June 2011, IEEE Computer Society, 2011, pp. 1633–1640.

[82] Marius Leordeanu and Martial Hebert, *A Spectral Technique for Correspondence Problems Using Pairwise Constraints*, in 10th IEEE International Conference on Computer Vision (ICCV 2005), 17-20 October 2005, Beijing, China, IEEE Computer Society, 2005, pp. 1482–1489.

[83] Marius Leordeanu, Andrei Zanfir, and Cristian Sminchisescu, *Semi-supervised learning and optimization for hypergraph matching*, in IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011, Dimitris N. Metaxas, Long Quan, Alberto Sanfeliu, and Luc J. Van Gool, eds., IEEE, 2011, pp. 2274–2281.

[84] Yangyan Li, Xiaokun Wu, Yiorgos Chrysanthou, Andrei Sharf, Daniel Cohen-Or, and Niloy J. Mitra, *GlobFit: consistently fitting primitives by discovering global relations*, ACM Transactions on Graphics (TOG), 30 (2011), p. 52.

[85] David G. Lowe, *Distinctive Image Features from Scale-Invariant Keypoints*, International Journal of Computer Vision (IJCV), 60 (2004), pp. 91–110.

[86] Mona Mahmoudi and Guillermo Sapiro, *Three-dimensional point cloud recognition via distributions of geometric distances*, Graphical Models, 71 (2009), pp. 22–31.

[87] George J. Mamic and Mohammed Bennamoun, *Representation and Recognition of 3D Free-Form Objects*, Digital Signal Processing, 12 (2002), pp. 47–76.

[88] Bogdan Matei, Ying Shan, Harpreet S. Sawhney, Yi Tan, Rakesh Kumar, Daniel F. Huber, and Martial Hebert, *Rapid Object Indexing Using Locality*

*Sensitive Hashing and Joint 3D-Signature Space Estimation*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 28 (2006), pp. 1111–1126.

[89] Manus McElhone, Jörg Stückler, and Sven Behnke, *Joint detection and pose tracking of multi-resolution surfel models in RGB-D*, in 2013 European Conference on Mobile Robots, Barcelona, Catalonia, Spain, September 25-27, 2013, IEEE, 2013, pp. 131–137.

[90] Donald Meagher, *Geometric modeling using octree encoding*, Computer Graphics and Image Processing, 19 (1982), pp. 129–147.

[91] Ajmal S. Mian, Mohammed Bennamoun, and Robyn A. Owens, *Automatic Correspondence for 3d Modeling: an Extensive Review*, International Journal of Shape Modeling, 11 (2005), pp. 253–291.

[92] Ajmal S. Mian, Mohammed Bennamoun, and Robyn A. Owens, *Three-Dimensional Model-Based Object Recognition and Segmentation in Cluttered Scenes*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 28 (2006), pp. 1584–1601.

[93] Ajmal S. Mian, Mohammed Bennamoun, and Robyn A. Owens, *On the Repeatability and Quality of Keypoints for Local Feature-based 3D Object Retrieval from Cluttered Scenes*, International Journal of Computer Vision (IJCV), 89 (2010), pp. 348–361.

[94] Lynette I. Millett and Samuel H. Fuller, *The Future of Computing Performance: Game Over or Next Level?*, National Academies Press, Washington, DC, 2011.

[95] Maher Moakher, *Means and Averaging in the Group of Rotations*, SIAM Journal on Matrix Analysis and Applications, 24 (2002), pp. 1–16.

[96] Thomas Morwald, Andreas Richtsfeld, Johann Prankl, Michael Zillich, and Markus Vincze, *Geometric data abstraction using B-splines for range image segmentation*, in 2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013, IEEE, 2013, pp. 148–153.

[97] Frederick Mosteller and John Wilder Tukey, *Data analysis and regression: a second course in statistics.*, Addison-Wesley Series in Behavioral Science: Quantitative Methods, (1977).

[98] David M. Mount and Sunil Arya, *ANN: A Library for Approximate Nearest Neighbor Searching*. `https://www.cs.umd.edu/~mount/ANN/`.

[99] Marius Muja and David G. Lowe, *Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration*, in VISAPP 2009 - Proceedings of the

Fourth International Conference on Computer Vision Theory and Applications, Lisboa, Portugal, February 5-8, 2009 - Volume 1, Alpesh Ranchordas and Helder Araújo, eds., INSTICC Press, 2009, pp. 331–340.

[100] MVTec Software GmbH, *HALCON - the power of machine vision*. `http://halcon.com`.

[101] Andriy Myronenko and Xubo B. Song, *Point Set Registration: Coherent Point Drift*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 32 (2010), pp. 2262–2275.

[102] Duc Dung Nguyen, Jae Pil Ko, and Jae Wook Jeon, *Determination of 3D object pose in point cloud with CAD model*, in 21st Korea-Japan Joint Workshop on Frontiers of Computer Vision, FCV 2015, Mokpo, South Korea, January 28-30, 2015, Soon-Young Park, Hironobu Fujiyoshi, Kunihito Kato, Hongbin Zha, Chil-Woo Lee, and Kang-Hyun Jo, eds., IEEE, 2015, pp. 1–6.

[103] Andreas Nüchter, Kai Lingemann, and Joachim Hertzberg, *Cached k-d tree search for ICP algorithms*, in Sixth International Conference on 3-D Digital Imaging and Modeling, 3DIM 2007, 21-23 August 2007, Montreal, Quebec, Canada, IEEE Computer Society, 2007, pp. 419–426.

[104] Abdul Nurunnabi, David Belton, and Geoff A. W. West, *Robust Segmentation in Laser Scanning 3D Point Cloud Data*, in 2012 International Conference on Digital Image Computing Techniques and Applications, DICTA 2012, Fremantle, Australia, December 3-5, 2012, IEEE, 2012, pp. 1–8.

[105] Bastian Oehler, Jörg Stückler, Jochen Welle, Dirk Schulz, and Sven Behnke, *Efficient Multi-resolution Plane Segmentation of 3D Point Clouds*, in Intelligent Robotics and Applications - 4th International Conference, ICIRA 2011, Aachen, Germany, December 6-8, 2011, Proceedings, Part II, Sabina Jeschke, Honghai Liu, and Daniel Schilberg, eds., vol. 7102 of Lecture Notes in Computer Science, Springer, 2011, pp. 145–156.

[106] Clark F. Olson and Daniel P. Huttenlocher, *Automatic target recognition by matching oriented edge pixels*, IEEE Transactions on Image Processing, 6 (1997), pp. 103–113.

[107] Jens Overby, Lars Bodum, Erik Kjems, and PM Iisoe, *Automatic 3D building reconstruction from airborne laser scanning and cadastral data using Hough transform*, The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences (IAPRS), 35 (2004), pp. 296–301.

[108] In Kyu Park, Marcel Germann, Michael D. Breitenstein, and Hanspeter Pfister, *Fast and automatic object pose estimation for range images on the GPU*, Machine Vision and Applications (MVA), 21 (2010), pp. 749–766.

[109] Georgios Passalis, Ioannis A. Kakadiaris, and Theoharis Theoharis, *Intraclass Retrieval of Nonrigid 3D Objects: Application to Face Recognition*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 29 (2007), pp. 218–229.

[110] T. Rabbani and F. Van Den Heuvel, *Efficient hough transform for automatic detection of cylinders in point clouds*, in Proceedings of the 11th Annual Conference of the Advanced School for Computing and Imaging (ASCI05), vol. 3, 2005, pp. 60–65.

[111] Narayan S. Raja and Anil K. Jain, *Recognizing geons from superquadrics fitted to range data*, Image and Vision Computing, 10 (1992), pp. 179–190.

[112] Olinde Rodrigues, *Des lois géométriques qui régissent les déplacements d'un système solide dans léspace, et de la variation des coordonnées provenant de ces déplacements considérés indépendamment des causes qui peuvent les produire*, Journal de Mathématiques Pures et Appliquées, 5 (1840), pp. 380–440.

[113] Salvador Ruiz-Correa, Linda G. Shapiro, and Marina Meila, *A New Paradigm for Recognizing 3-D Object Shapes from Range Data*, in 9th IEEE International Conference on Computer Vision (ICCV 2003), 14-17 October 2003, Nice, France, IEEE Computer Society, 2003, pp. 1126–1133.

[114] Szymon Rusinkiewicz and Marc Levoy, *Efficient Variants of the ICP Algorithm*, in 3rd International Conference on 3D Digital Imaging and Modeling (3DIM 2001), 28 May - 1 June 2001, Quebec City, Canada, IEEE Computer Society, 2001, pp. 145–152.

[115] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz, *Fast Point Feature Histograms (FPFH) for 3D registration*, in 2009 IEEE International Conference on Robotics and Automation, ICRA 2009, Kobe, Japan, May 12-17, 2009, IEEE, 2009, pp. 3212–3217.

[116] Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz, *Aligning point cloud views using persistent feature histograms*, in 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, September 22-26, 2008, Acropolis Convention Center, Nice, France, IEEE, 2008, pp. 3384–3391.

[117] Radu Bogdan Rusu, Gary R. Bradski, Romain Thibaux, and John M. Hsu, *Fast 3D recognition and pose using the Viewpoint Feature Histogram*, in 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 18-22, 2010, Taipei, Taiwan, IEEE, 2010, pp. 2155–2162.

[118] Renato F. Salas-Moreno, Richard A. Newcombe, Hauke Strasdat, Paul H. J. Kelly, and Andrew J. Davison, *SLAM++: Simultaneous Localisation*

*and Mapping at the Level of Objects*, in 2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, June 23-28, 2013, IEEE, 2013, pp. 1352–1359.

[119] Samuele Salti, Federico Tombari, and Luigi di Stefano, *SHOT: Unique signatures of histograms for surface and texture description*, Computer Vision and Image Understanding, 125 (2014), pp. 251–264.

[120] Hanan Samet, *Foundations of Multidimensional And Metric Data Structures*, Morgan Kaufmann, 2006.

[121] Peter Schatte, *Computing the Angle between Vectors*, Computing, 63 (1999), pp. 93–96.

[122] EJ Schlossmacher, *An iterative technique for absolute deviations curve fitting*, Journal of the American Statistical Association, 68 (1973), pp. 857–859.

[123] Ruwen Schnabel, Roland Wahl, and Reinhard Klein, *Efficient RANSAC for Point-Cloud Shape Detection*, Computer Graphics Forum, 26 (2007), pp. 214–226.

[124] Dino Schweitzer, Andrew Glassner, and Mike Keeler, eds., *Proceedings of the 21th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1994, Orlando, FL, USA, July 24-29, 1994*, ACM, 1994.

[125] Thomas W. Sederberg and Scott R. Parry, *Free-form deformation of solid geometric models*, in Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1986, Dallas, Texas, USA, August 18-22, 1986, David C. Evans and Russell J. Athay, eds., ACM, 1986, pp. 151–160.

[126] Ying Shan, Bogdan Matei, Harpreet S. Sawhney, Rakesh Kumar, Daniel F. Huber, and Martial Hebert, *Linear Model Hashing and Batch RANSAC for Rapid and Accurate Object Recognition*, in IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2, 2004, pp. 121–128.

[127] Øystein Skotheim, Morten Lind, Pål Ystgaard, and Sigurd Aksnes Fjerdingen, *A flexible 3D object localization system for industrial part handling*, in 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012, IEEE, 2012, pp. 3326–3333.

[128] Øystein Skotheim, Jens T Thielemann, Asbjørn Berge, and Arne Sommerfelt, *Robust 3D object localization and pose estimation for random bin picking with the 3DMaMa algorithm*, in IS&T/SPIE Electronic Imaging, International Society for Optics and Photonics, 2010, pp. 75260E–75260E.

[129] Nikhil Somani, Caixia Cai, Alexander Clifford Perzylo, Markus Rickert, and Alois Knoll, *Object Recognition Using Constraints from Primitive Shape Matching*, in Advances in Visual Computing - 10th International Symposium, ISVC 2014, Las Vegas, NV, USA, December 8-10, 2014, Proceedings, Part I, George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Ryan McMahan, Jason Jerald, Hui Zhang, Steven M. Drucker, Chandra Kambhamettu, Maha El Choubassi, Zhigang Deng, and Mark Carlson, eds., vol. 8887 of Lecture Notes in Computer Science, Springer, 2014, pp. 783–792.

[130] Bastian Steder, Radu Bogdan Rusu, Kurt Konolige, and Wolfram Burgard, *Point feature extraction on 3D range scans taking into account object boundaries*, in IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011, IEEE, 2011, pp. 2601–2608.

[131] Carsten Steger, *Occlusion, Clutter, and Illumination Invariant Object Recognition*, in The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences (IAPRS), 2002, pp. 345–350.

[132] Carsten Steger, Markus Ulrich, and Christian Wiedemann, *Machine Vision Algorithms and Applications*, Wiley-VCH Verlag GmbH & Co. KGaA, 1. auflage ed., Nov. 2007.

[133] Fridtjof Stein and Gérard G. Medioni, *Structural Indexing: Efficient 3-D Object Recognition*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 14 (1992), pp. 125–145.

[134] Charles V. Stewart, *Robust Parameter Estimation in Computer Vision*, SIAM Review, 41 (1999), pp. 513–537.

[135] Stefan Stiene, Kai Lingemann, Andreas Nüchter, and Joachim Hertzberg, *Contour-Based Object Detection in Range Images*, in 3rd International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT 2006), 14-16 June 2006, Chapel Hill, North Carolina, USA, IEEE Computer Society, 2006, pp. 168–175.

[136] George C. Stockman, *Object recognition and localization via pose clustering*, Computer Vision, Graphics, and Image Processing, 40 (1987), pp. 361–387.

[137] John Stuelpnagel, *On the parametrization of the three-dimensional rotation group*, SIAM review, 6 (1964), pp. 422–430.

[138] Min Sun, Gary R. Bradski, Bing-Xin Xu, and Silvio Savarese, *Depth-Encoded Hough Voting for Joint Object Detection and Shape Recovery*, in Computer Vision - ECCV 2010 - 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part V, Kostas Daniilidis, Petros Maragos, and Nikos Paragios, eds., vol. 6315 of Lecture Notes in Computer Science, Springer, 2010, pp. 658–671.

[139] Yiyong Sun, Joon Ki Paik, Andreas F. Koschan, David L. Page, and Mongi A. Abidi, *Point fingerprint: A new 3-D object representation scheme*, IEEE Transactions on Systems, Man, and Cybernetics, Part B, 33 (2003), pp. 712–717.

[140] Babak Taati and Michael A. Greenspan, *Local shape descriptor selection for object recognition in range data*, Computer Vision and Image Understanding, 115 (2011), pp. 681–694.

[141] Fayez Tarsha-Kurdi, Tania Landes, and Pierre Grussenmeyer, *Hough-transform and extended ransac algorithms for automatic detection of 3d building roof planes from lidar data*, in ISPRS Workshop on Laser Scanning 2007 and SilviLaser 2007, vol. 36, 2007, pp. 407–412.

[142] Jeanette A. Thomas, *Echolocation in Bats and Dolphins*, University of Chicago Press, 2004.

[143] Geoffrey Timmins, *The last shift: the decline of handloom weaving in nineteenth-century Lancashire*, Manchester University Press ; Distributed exclusively in the USA and Canada by St. Martin's Press, Manchester; New York, 1993.

[144] Federico Tombari, Samuele Salti, and Luigi di Stefano, *Unique Signatures of Histograms for Local Surface Description*, in Computer Vision - ECCV 2010, 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part III, Kostas Daniilidis, Petros Maragos, and Nikos Paragios, eds., vol. 6313 of Lecture Notes in Computer Science, Springer, 2010, pp. 356–369.

[145] Federico Tombari, Samuele Salti, and Luigi di Stefano, *A combined texture-shape descriptor for enhanced 3D feature matching*, in 18th IEEE International Conference on Image Processing, ICIP 2011, Brussels, Belgium, September 11-14, 2011, Benoît Macq and Peter Schelkens, eds., IEEE, 2011, pp. 809–812.

[146] Aksel Andreas Transeth, Øystein Skotheim, Henrik Schumann-Olsen, Gorm Johansen, Jens T Thielemann, and Erik Kyrkjebø, *A robotic concept for remote maintenance operations: A robust 3D object detection and pose estimation method and a novel robot tool.*, in IROS, 2010, pp. 5099–5106.

[147] Oncel Tuzel, Ming-Yu Liu, Yuichi Taguchi, and Arvind Raghunathan, *Learning to Rank 3D Features*, in Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I, David J. Fleet, Tomás Pajdla, Bernt Schiele, and Tinne Tuytelaars, eds., vol. 8689 of Lecture Notes in Computer Science, Springer, 2014, pp. 520–535.

[148] Markus Ulrich, Carsten Steger, and Albert Baumgartner, *Real-time object recognition using a modified generalized Hough transform*, Pattern Recognition, 36 (2003), pp. 2557–2570.

[149] George Vosselman and Sander Dijkman, *3D building model reconstruction from point clouds and ground plans*, The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences (IAPRS), 34 (2001), pp. 37–44.

[150] Eric Wahl, Ulrich Hillenbrand, and Gerd Hirzinger, *Surflet-Pair-Relation Histograms: A Statistical 3D-Shape Representation for Rapid Classification*, in 4th International Conference on 3D Digital Imaging and Modeling (3DIM 2003), 6-10 October 2003, Banff, Canada, IEEE Computer Society, 2003, pp. 474–482.

[151] Andrew Willis and Beibei Zhou, *Ridge Walking for 3D Surface Segmentation*, in Proceedings of Fifth Asian Conference on Computer Vision, Paris, France, 2010.

[152] Simon Winkelbach, Sven Molkenstruck, and Friedrich M. Wahl, *Low-Cost Laser Range Scanner and Fast Surface Registration Approach*, in Pattern Recognition, 28th DAGM Symposium, Berlin, Germany, September 12-14, 2006, Proceedings, Katrin Franke, Klaus-Robert Müller, Bertram Nickolay, and Ralf Schäfer, eds., vol. 4174 of Lecture Notes in Computer Science, Springer, 2006, pp. 718–728.

[153] Changchang Wu, Brian Clipp, Xiaowei Li, Jan-Michael Frahm, and Marc Pollefeys, *3D model matching with Viewpoint-Invariant Patches (VIP)*, in 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA, IEEE Computer Society, 2008.

[154] Sameh M. Yamany and Aly A. Farag, *Surfacing Signatures: An Orientation Independent Free-Form Surface Representation Scheme for the Purpose of Objects Registration and Matching*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 24 (2002), pp. 1105–1120.

[155] Ping Yan and Kevin W. Bowyer, *A fast algorithm for ICP-based 3D shape biometrics*, Computer Vision and Image Understanding, 107 (2007), pp. 195–202.

[156] T. Zaharia and F. Prêteux, *Hough transform-based 3D mesh retrieval*, in Proceedings of the SPIE Conf. 4476 on Vision Geometry X, 2001, pp. 175–185.

[157] Ron Zass and Amnon Shashua, *Probabilistic graph and hypergraph matching*, in 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA, IEEE Computer Society, 2008.

[158] Zhengyou Zhang, *Iterative point matching for registration of free-form curves and surfaces*, International Journal of Computer Vision (IJCV), 13 (1994), pp. 119–152.