An abstract graphic on the left side of the page, featuring a vertical gradient from light orange to light blue. It contains several large, stylized, brown geometric shapes that resemble arrows or abstract letters, pointing in various directions.

Learning Strategies to Select Point Cloud Descriptors for Large-Scale 3-D Object Classification

Jens Garstka
2016

© 2016 Jens Garstka

Editor:	Dean of the Department of Mathematics and Computer Science
Type and Print:	FernUniversität in Hagen
Distribution:	http://deposit.fernuni-hagen.de/view/departments/miresearchreports.html

Learning Strategies to Select Point Cloud Descriptors for Large-Scale 3-D Object Classification

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

der Fakultät Mathematik und Informatik
der FernUniversität in Hagen

vorgelegt von

Dipl.-Inf. Jens Garstka
geb. in Bochum

Hagen, 21. September 2015

GUTACHTERIN UND GUTACHTER:

Univ.-Prof. Dr. Gabriele Peters

Univ.-Prof. Dr. Heinrich Müller

Zusammenfassung

Maschinelles Lernen ist ein weites Feld der Informatik, welches sich mit Algorithmen beschäftigt, die in der Lage sind, aus Daten zu lernen. Das Ziel hierbei ist nicht nur die Erzeugung von Wissen. Es existiert zudem ein wachsender Bedarf an selbst lernenden, intelligenten Systemen, welche in der Lage sind, auch unter veränderlichen Bedingungen zu agieren. Eine allgemein übliche Vorgehensweise, den Bereich des maschinellen Lernens in Teilbereiche zu strukturieren, besteht in einer Einteilung in Methoden des überwachten, unüberwachten und bestärkenden Lernens (Reinforcement Learning). Die letztgenannten Methoden eignen sich insbesondere für autonome Agenten, die lernen, während sie mit der Umgebung interagieren. Reinforcement Learning kann als eben diese langfristige Interaktion eines lernenden Agenten und einer sich verändernden Umgebung verstanden werden, bei welcher das der Umgebung zugrunde liegende Modell für den Agenten verborgen bleibt. Der Agent lernt daher ausschließlich aufgrund der Beobachtung der Umgebung. Einerseits wird Reinforcement Learning als Hilfsmittel verwendet, mit welchem die theoretischen Grundlagen des agentenbasierten Lernens erforscht werden können. Andererseits existieren zahlreiche praktische Implementierungen, wie zum Beispiel autonome Roboter oder industrielle Anlagen, bei welchen sich die Systeme durch Erfahrung selbst verbessern. Als weiteres Anwendungsfeld des Reinforcement Learnings gelten zudem kombinatorische Suchprobleme.

Während sich die meisten praktischen Ansätze, Reinforcement Learning zur Lösung solcher Suchprobleme zu verwenden, auf den Bereich der Computerspiele beschränken, ist eine Motivation dieser Arbeit zu zeigen, dass Reinforcement Learning auch zur Lösung solcher Suchprobleme in einem Forschungsbereich des maschinellen Sehens verwendet werden kann, der aktuell unter anderem aufgrund großer, in der Praxis anfallender Datenmengen vor noch ungelösten Herausforderungen steht. Der Fokus dieser Arbeit liegt hier auf einem bedeutenden Teil des maschinellen Sehens, der Klassifikation von 3D-Punktwolken. Von einer zuverlässigen und effizienten Klassifikation von 3D-Punktwolken würde ein großer Bereich unterschiedlicher Anwendungsfelder profitieren, wie beispielsweise ein au-

tomatisiertes Szenenverständnis, die Navigation in unbekanntem Terrain oder das Greifen und Manipulieren von Objekten in der Robotik.

Die meisten aktuellen Systeme zur Klassifikation von 3D-Punktwolken verwenden Algorithmen, die sogenannte 3D-Merkmalsbeschreibungen als niedrig-dimensionale Beschreibungen der gesamten 3D-Punktwolke oder lokaler Teilbereiche derselben bestimmen. Einige dieser Algorithmen sind schnell aber ungenau, andere erfordern individuell abgestimmte Werte für zahlreiche Parameter oder benötigen sehr viel Rechenzeit für die Berechnung der 3D-Merkmalsbeschreibungen und deren Vergleich. Es existiert aber bislang kein Verfahren, welches in jeder Situation zufriedenstellend funktioniert.

Dies legt eine Kombination unterschiedlicher Verfahren zur Berechnung von 3D-Merkmalsbeschreibungen nahe. In dieser Arbeit wird der Nutzen einer sukzessiven Anwendung verschiedener, aktueller Techniken der Merkmalsbeschreibung für die 3D-Objektklassifikation untersucht. Dies kann als kombinatorisches Suchproblem betrachtet werden, welches sich aus der Fragestellung ergibt, welche Verfahren zur Klassifikation von 3D-Punktwolken verwendet werden und in welcher Reihenfolge selbige angewandt werden sollen. Um dies in ein Lernproblem für einen Reinforcement Learning Ansatz zu übertragen und bekannte Einschränkungen diesbezüglich zu überwinden, geht diese Arbeit unter anderem auch auf fundamentale Hindernisse wie einen zu großen Zustandsraum ein. Zudem wird gezeigt, wie Online-Lernmethoden des Reinforcement Learnings zugunsten der Anpassungsfähigkeit des Systems genutzt werden können, so dass zum Beispiel neu hinzugefügte Verfahren für 3D-Merkmalsbeschreibungen ohne Einschränkungen in das laufende System integriert werden können. Abschließend wird gezeigt, dass der vorgestellte Ansatz in der Lage ist, durch Kombination einzelner Verfahren bessere Klassifikationsergebnisse zu erreichen, als es mit einem einzelnen Verfahren möglich ist. Anhand des Anwendungsbeispiels der Klassifikation von 3D-Punktwolken wird gezeigt, dass selbst erlernte Kombinationen unterschiedlicher Lösungsansätze für ein und dieselbe Problemstellung das endgültige Ergebnis zu verbessern vermögen. Neben diesem eher theoretischen, für die Grundlagen- und maschinellen Lernens relevanten Erkenntnisgewinn besteht der Hauptbeitrag dieser Arbeit in ihrem praktischen Nutzen, indem gezeigt wird, dass die vorgeschlagene Vorgehensweise, 3D-Objektklassifikation mithilfe eines Reinforcement Learning Ansatzes adaptiv zu gestalten, einen möglichen Lösungsansatz für aktuell ungelöste Probleme des Computersehens darstellt.

Preface

Machine learning is a wide field of computer science, which is concerned with algorithms that are able to learn from data. Not only the acquisition of knowledge is its main goal. Rather there is a growing need for self-learning, intelligent systems, that are able to act under dynamic conditions in their environment. One common way to structure the field of machine learning is the classification into supervised, unsupervised, and reinforcement learning methods. The latter are especially suited to create autonomous agents that learn while interacting with the environment. The reinforcement learning problems can be outlined as long term interaction between a learning agent and a dynamic environment, where the underlying model is not visible to the agent, so that the agent learns only from the observations of the environment. On the one hand reinforcement learning serves as a theoretical tool for studying the principles of agents learning to act. On the other hand there are many examples of practical implementations, e. g., autonomous robots or industrial systems that improve themselves with experience. Other fields of application are combinatorial search problems.

While the most practical cases of such search problems in the context of reinforcement learning are restricted solely to the games context, one motivation of this thesis is to show, that reinforcement learning can also be used to solve such search problems at the frontier of computer vision research. This will be done with special focus on an important part of computer vision that currently faces unsolved challenges, for example because of the large amount of data that accumulate in practical applications: the classification of 3-D point clouds. A wide range of applications such as scene understanding, navigation, or applications in robotics like grasping or scene manipulation would benefit from a reliable and efficient classification of 3-D point clouds.

Most of the current 3-D classification systems use so called 3-D feature descriptors to compute lower dimensional descriptions of the entire 3-D point cloud or of local parts of it. While some of them are fast but inaccurate, others have

a large number of parameters or their computational costs of calculation and comparison are high. However, there are no 3-D feature descriptors which work satisfying in all situations.

This suggests the combination of different feature descriptors. In this thesis the benefit of a successive application of state-of-the-art 3-D feature descriptors for the classification of 3-D objects is investigated. This can be regarded as a combinatorial search problem arising from the question which feature descriptors should be used to classify an object and in which order. To be able to translate this into a reinforcement learning problem and to overcome its known limitations, this thesis addresses fundamental obstacles such as a possibly large state space and shows how on-line learning can lead to an adaptive system, that is, for example, able to adaptively integrate new feature descriptors into its learned classification strategy. Finally, it is shown that the proposed approach of combining several algorithms leads to better classification results than the application of one single algorithm alone. Thus, through the example of 3-D point cloud classification, the thesis shows how a self-learned combination of different approaches to the same problem can improve the final result. Besides this more theoretically relevant insight the main contribution of this work consists on its practical value, namely in the demonstration that the proposed proceeding of an adaptive 3-D object classification via reinforcement learning is a possible approach to finding a solution for a current challenge in computer vision.

Contents

1	Introduction	1
1.1	From Object Classification to Scene Understanding	1
1.1.1	Methods of Object Recognition and Classification	2
1.1.2	The 3rd Dimension	3
1.2	Contribution of the Thesis	3
1.3	Organization of the Thesis	4
1.4	Concepts and Terminology	5
1.4.1	Three Dimensions	5
1.4.2	Keypoints and Descriptions	5
2	Related Work	7
2.1	Recognition and Classification Pipelines	8
2.2	Keypoint Detection	10
2.3	3-D Object Description	10
2.3.1	Local 3-D Feature Descriptors	11
2.3.2	Global 3-D Descriptors	11
2.3.3	Hybrid Feature Descriptors	13
2.4	Reinforcement Learning	15

3	Keypoint Detection	17
3.1	Keypoint Detection Algorithms	18
3.1.1	Multi-scale Feature Extraction on Point-Sampled Surfaces .	18
3.1.2	Integral Volume Descriptor	19
3.1.3	Selection by the Smallest Eigenvalues of Local Regions . . .	22
3.1.4	Local Surface Patches	23
3.1.5	Thrift	23
3.1.6	Multi-Scale Interest Regions from Point Clouds	25
3.1.7	2.5-D SIFT	26
3.1.8	Intrinsic Shape Signature	27
3.1.9	A Keypoint Quality Measure using Surface Variations . . .	27
3.1.10	Scale-Space Surface Analysis in Depth Images	28
3.1.11	Point Feature Extraction on 3D Range Scans	29
3.1.12	Interest Points of Local Surface Entropy	31
3.2	Performance Evaluation	33
4	3-D Object Description	35
4.1	Local Feature Descriptors	35
4.1.1	Spin Images	36
4.1.2	3-D Shape Context	37
4.1.3	Local Surface Patches	38
4.1.4	Thrift	39
4.1.5	Point Feature Histogram	40
4.1.6	Fast Point Feature Histogram	42
4.1.7	2.5-D SIFT	43
4.1.8	Intrinsic Shape Signatures	44
4.1.9	Shape Index SIFT	46
4.1.10	Surface Fitting with a Uniform Lattice	46

4.1.11	NARF	47
4.1.12	Signatures of Histograms of Orientations	48
4.1.13	Unique Shape Context	50
4.1.14	SURE	51
4.1.15	Histogram of Oriented Normal Vectors	51
4.2	Performance Evaluation of Feature Descriptors	52
4.3	Bag of Features	53
5	Support Vector Machines	55
5.1	Linear Separable Data	55
5.2	Nonlinear Separation and the Kernel-Trick	57
5.3	Kernel Functions	58
6	Reinforcement Learning	61
6.1	Markov Decision Process	62
6.2	Reward and Return	63
6.3	Value Functions and Policy	64
6.4	Optimal Value Functions	64
6.5	Learning Policies	65
6.6	Exploration-Exploitation Dilemma	67
7	Survey on Methods	71
7.1	Description of the Problem	71
7.2	Approaches	72
7.2.1	The Basic Classification Pipeline	72
7.2.2	Fusion with Reinforcement Learning	78
7.2.3	Differentiation of the First State	82
7.2.4	Adaptive Learning	83
7.3	Evaluation Methods	84

7.3.1	Experimental Datasets	84
7.3.2	Computation Times	87
7.3.3	Time Limit	88
7.3.4	Handling the Data	88
8	Baseline Method	91
8.1	Preprocessing	91
8.1.1	Cleaning up Raw Data	91
8.1.2	Approximation of the Point Cloud Resolution	92
8.2	Keypoint Detection	98
8.2.1	Number of Keypoints	99
8.2.2	Computation Time	100
8.2.3	Interim conclusion on Keypoints	101
8.3	Local 3-D Feature Descriptors	102
8.3.1	The algorithms	102
8.3.2	Computation Times	108
8.3.3	Interim conclusion on Local 3-D Feature Descriptors	110
8.4	Bag of Features	110
8.4.1	Vocabulary Construction	110
8.4.2	Computation of Histograms	112
8.5	Training and Classification Results	112
8.5.1	Optimal Training Parameters – a Single Case	112
8.5.2	Optimal Training Parameters – Comparison	117
8.5.3	Local 3-D Feature Descriptors	121
8.5.4	Precision and Recall	124
8.5.5	Increasing the Limit of the Prediction Values	125
8.5.6	Reducing the Number of Classes	127
8.5.7	Interim Conclusion on Training and Classification Results	128

8.6	Conclusion	131
9	Adaptive 3-D Object Classification	133
9.1	Fusion with Reinforcement Learning	133
9.1.1	Results for all Object Classes – Different Limits	136
9.1.2	Results for 10 Object Classes – Different Limits	139
9.1.3	Results for all Object Classes – Unique Limits	141
9.1.4	Results for 10 Object Classes – Unique Limits	142
9.1.5	Interim Conclusion on Fusion with Reinforcement Learning	145
9.2	Differentiation of the First State	145
9.2.1	Results	146
9.2.2	Conclusion	149
9.3	Adaptive Learning	149
9.3.1	Results	150
9.3.2	Conclusion	153
10	Comparative Evaluation of Results	155
10.1	Initial Situation	155
10.2	Classification Results	156
10.3	Adaptivity	158
10.4	Computation Time	158
11	Conclusion and Outlook	159
11.1	Conclusion	159
11.2	Future Work	160

A Software	163
A.1 System	163
A.2 Development Environment	163
A.3 Libraries	164
A.3.1 VTK	164
A.3.2 OpenCV	165
A.3.3 PCL	165
A.3.4 ROOT	166
B Datasets	167
B.1 Stanford 3-D Scanning Repository	167
B.2 3-D Shape Retrieval Contest 2013	169
B.3 RGB-D Object Dataset	171
C Extended Results	177
C.1 Approximation of the Mesh Resolution	177
C.2 Parameter Optimization	184
C.2.1 3-D Shape Context	184
C.2.2 Fast Point Feature Histogram	187
C.2.3 Point Feature Histogram	190
C.2.4 Signature of Histograms of Orientations	193
C.2.5 Spin Images	196
C.2.6 Unique Shape Context	199
List of figures	206
Bibliography	218

Summary of Notation

Nomenclature

The nomenclature which is generally followed throughout this work concerning equations and formulas is captured by the following list of examples.

Common Notation

a	values and scalar functions
A	random variables and major algorithm variables
$\mathcal{A}$	sets

Vector Spaces

$\mathbf{a}$	vectors of $\mathbb{R}^2$
$\mathbf{a}$	vectors of $\mathbb{R}^n$ with $n \geq 3$
$\mathbf{A}$	matrices
H	histograms
$\ \mathbf{a}\ $	l^2 -norm
$\langle \mathbf{a}, \mathbf{b} \rangle$	inner product or scalar product

Reinforcement Learning

s	state
a	action
$\mathcal{S}$	set of states
$\mathcal{A}(s)$	set of actions possible in state s
π	policy
$r(s, a, s')$	immediate reward on transition s to s' under action a
$V_{\pi}(s)$	state-value function – possible future reward in state s following policy π
$Q_{\pi}(s, a)$	action-value function – possible future reward in state s when taking action a and following policy π

Abbreviations

The following abbreviations are used to dispense with the unwieldy terms, e. g., in a context of a table.

Keypoint Detectors

ISS	intrinsic shape signatures (see Section 3.1.8)
-----	--

Local 3-D Feature Descriptors

3DSC	3-D shape context (see Section 4.1.2)
FPFH	fast point feature histogram (see Section 4.1.6)
PFH	point feature histogram (see Section 4.1.5)
SHOT	signature of histograms of orientations (see Section 4.1.12)
SI	spin images (see Section 4.1.1)
USC	unique shape context (see Section 4.1.13)

Other Algorithms

SVM	support vector machine (see Section 5)
FFT	fast Fourier transform
PCA	principal component analysis

1

Introduction

The time where computers have stepped out of the stage of development where their only task was to assist people to solve merely tedious and error-prone calculations, is long ago. Nowadays, computers are capable to mimic human activities which often require a certain kind of intelligence. With the increasing computational capacity and the declining costs of such systems, their field of application is also constantly expanding. One of these new technologies with great potential is to capture the environment using a 3-D sensor in order to analyze and interpret the content of a scene with the final goal of *scene understanding*.

The benefits of scene understanding are manifold. Autonomous robots and interactive systems will profit from it, e. g., for the purpose of orienting themselves in unfamiliar environments, for the detection of certain objects, or for their manipulation. In this context, autonomous vehicles should not remain unmentioned, since they combine many of these requirements. Further fields of application are, for example, augmented reality in which virtual objects are integrated into real scenes, or environmental monitoring in the context of smart homes or surveillance.

1.1 From Object Classification to Scene Understanding

A simple form of object recognition identifies a single isolated object in one image or a sequence of images, i. e., a video. This remains the most common form of object recognition today. If the searched object has a unique texturing, this type of object recognition has already reached a level of maturity that facilitates its

active usage in many areas. Certainly the most popular example for this is the traffic sign recognition in modern cars.

The next challenge is an abstraction of the objects that should be recognized within a scene. Only in this way it is possible to recognize the plurality of different and similar objects under varying conditions. For this classification task of multiple objects in various scenes, the individual objects have to be distinguished and separated from unimportant elements such as the background. Classification of 3-D objects is the problem this thesis deals with. Finally, in order to understand the meaning of a scene, the detected objects have to be brought into relationship, which is beyond the scope of this thesis.

1.1.1 Methods of Object Recognition and Classification

There are many ways to classify objects by analysis of a wide variety of sensor data such as color images, radar signals, 3-D points clouds from rotating laser scanners, depth data from time-of-flight cameras. A significant break-through in this complex but important area of computer vision was brought by the development of scale and rotation invariant feature descriptors for color images over the last 15 years. These methods have become integral parts of computer vision and can be found in many application scenarios, e.g., in cameras with built-in face detection and tracking, in augmented reality applications, e.g., city guides which superimpose and align historical photos in live views, or as visual control in manufacturing processes, just to name a few examples. By the end of 2014 research teams from Google [97] and Stanford University [39] have demonstrated independently the capability of these 2-D methods in combination with artificial intelligence.

There are, however, a few situations where 2-D color images are not a sufficient source to classify objects. The methods mentioned above usually require color information or at least enough contrast between elements of a picture to classify the displayed objects. Accordingly, these methods work neither under difficult lighting conditions, e.g., in darkness nor on objects without appropriate color or brightness structures, e.g., a gray cup of coffee on a gray office table. A possibility to solve these problems or essentially decrease their disadvantage is to exploit the third dimension as additional source of information.

1.1.2 The 3rd Dimension

With the advent of new cheaply available depth sensors, the use of the third dimension became available to a rapidly growing group of people who use the three-dimensional (3-D) data in a wide range of 3-D applications. This gave rise to a number of new algorithms for object recognition and classification tasks on the basis of object representations in the form of 3-D point clouds in the last few years. However, in contrast to object recognition and classification based on color images, none of the current algorithms that utilize the 3-D point cloud representations of objects was able to facilitate the decisive breakthrough in the field of object classification, neither in terms of higher recognition and classification rates, nor in terms of lower computation times.

1.2 Contribution of the Thesis

Considering the aforementioned state of object recognition and classification utilizing 3-D point clouds, one can ask for the reasons why the description of objects in the form of 3-D point clouds does – despite using similar approaches as for color images – not facilitate improved classification and recognition results.

There are several reasons for this. On the one hand the processing of 3-D data is due to the continuous domain of the coordinates of 3-D points often computationally expensive. Additionally, the calculation effort is, in contrast to other 3-D object representations such as meshes, higher, because 3-D point clouds typically do not contain structural information such as edges or faces of a mesh. On the other hand the depth information provided by sensors is often affected by excessive noise, or it is quantized in a way that the depth value of the 3-D point is too vague to allow a high-quality description of a 3-D object.

One possibility would be to develop yet another algorithm to create descriptions based on 3-D point clouds. This algorithm would, however, be struggling with the same problems as all other state-of-the-art algorithms before. Since many of these algorithms are based on different concepts, the question arises whether a skillful combination of several algorithms can improve the detection and classification results.

The contribution of this thesis consists in the development of a reinforcement learning based adaptive selection of state-of-the-art descriptors and their succes-

sive application to 3-D point clouds. Furthermore, it investigates the capabilities if this approach to improve 3-D object classification results, especially in the context of large-scale applications with highly numerous and similar objects.

Starting point of this work is the precise treatment and analysis of existing 3-D point cloud classification techniques. Therefore, the functional principle of the most widely used classification pipeline is examined and all relevant components are analyzed. This begins with the selection of an algorithm for the detection of the so-called keypoints, continues with the inspection of numerous algorithms for the local description of 3-D point clouds and closes with an optimization of numerous parameters of the classification method used.

For all experiments, a large-scale data set of the University of Washington is used. The data set contains more than 200000 distinctive point clouds from 300 different objects which are organized in 51 object classes.

An important aspect of the approach in this thesis is its adaptivity, which is introduced by the reinforcement learning component. It allows for a dynamic change of the set of algorithms that are available to the classification system, while the system is on duty.

When algorithms are removed or new algorithms are added, the system is still able to integrate these changes without a temporary loss of its capability to classify objects. The thesis closes with a demonstration of the effectiveness of this concept.

1.3 Organization of the Thesis

The Chapters 2, 3, 4, 5, and 6 discuss related work of this thesis. While Chapter 2 gives a brief overview on the concepts used as foundation of this thesis, the subsequent chapters introduce the algorithms required for the basic classification pipeline and reinforcement learning in more detail.

Chapter 7 provides a detailed description of the problems discussed in this work as well as the description of the approaches to solve them. Chapter 8 includes the detailed analysis of the basic 3-D classification pipeline. This analysis is used to determine an optimized combination of the individual components of the classification pipeline, so that the pipeline delivers the best possible results.

Chapter 9 treats the results which can be achieved using the proposed classification approach based on reinforcement learning. Individual results are evaluated and discussed as part of the experiments in this chapter. Finally, Chapter 10 summarizes the individual results and joins them into an overall picture. Chapter 11 concludes this thesis with a summary and an outlook on future work.

1.4 Concepts and Terminology

1.4.1 Three Dimensions

Nowadays it is common practice to use digital cameras for capturing and recording images. These images are usually stored with the intensity values of the red, green and blue color channels (RGB color space) or with the raw sensor data in a 2-D matrix. The elements of this matrix are called *pixel*.

In contrast, there are several common representations of 3-D data. *Depth images* are 2-D matrices where the pixels contain projective or orthogonal distances to captured elements in space. The distances refer to the origin of the camera coordinate system or the image plane. *Point clouds* are defined as set of points $\mathbf{p}_i = (x_i, y_i, z_i)^\top \in \mathbb{R}^3$. The term “cloud” reflects the unorganized nature of the set and its spatial coherence.

If the points of a point cloud obtain a structure, e.g., *polygonal meshes*, the points of this mesh are often referred to as *vertices*. They either refer to a local or to a global reference frame. Meshes often contain additional structural information as the surface direction specified by *normal vectors*. Normal vectors are often determined for point clouds, too. In these cases the normal vectors can only be approximated using the adjacent points.

In addition, there are further 3-D formats such as *NURBS* and *splines* to describe a surface in a more mathematical way, but they play a subordinate role in context of this work.

1.4.2 Keypoints and Descriptions

To avoid the application of certain algorithms on all available points of a 3-D point cloud, a subset of points is determined which is representative of the entire

point cloud. This subset which is referred to as *keypoints* is used on all further calculations.

On the one hand keypoints can be selected by simply thinning out the point cloud. However, in the majority of cases the algorithms for keypoint detection are directly developed together with the algorithms which shall be executed subsequently. Thus, the selection of the keypoints can be adapted to the requirements of these algorithms.

Algorithms that create local 3-D feature descriptions of the neighborhood of a keypoint are the so-called *local 3-D feature description algorithms*. Sometimes this term may be shortened to *local 3-D feature descriptor* or just *feature descriptor*. The resulting *local 3-D feature descriptions* are also shortened to *local feature descriptions* or just *feature descriptions*.

In addition to the local 3-D feature descriptions, there are also global descriptions of a 3-D point cloud mentioned within the context of this work. In order to get a clear distinction between the local and global approaches, the term “feature” is never used in the context of a global description of a point cloud.

This chapter gives a brief overview of 3-D recognition and classification pipelines and provides an introduction of the important and typical parts of those. At a conceptual level, a 3-D classification pipeline is essentially based on four main components or steps. These four main steps correspond to the keypoint detection [72, 18, 19], the extraction of local feature descriptions [4, 27], a bag of words model, and support vector machines that are mainly used as a machine learning method for the classification task [91, 102, 12]. Figure 2.1 depicts these steps with a conceptual illustration of such a pipeline.

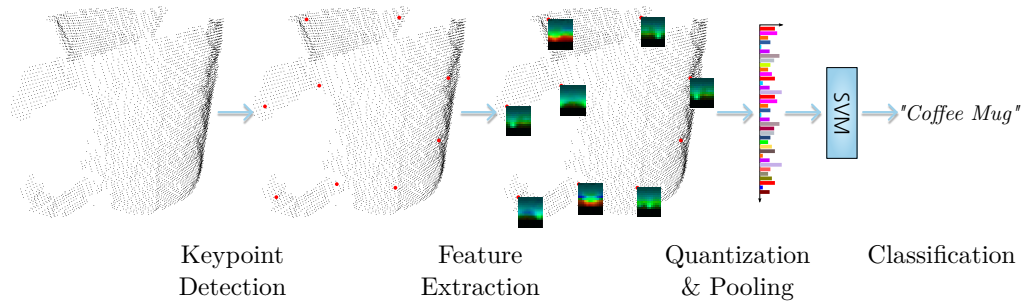


Figure 2.1: A conceptual illustration of a pipeline for 3-D object classification. It depicts a system based on local feature descriptors. The pipeline consists of four individual steps: the detection of keypoints, the computation of feature vectors for all keypoints, the quantization and pooling of all features (bag of words model), and the classification.

As already noted in the introduction, reinforcement learning is used as learning method to combine feature descriptor algorithms to sequences within the classification pipeline to thereby improve the classification results. Hence, an overview of reinforcement learning is subsequently given.

Each step, and, especially the keypoint detection and feature description algorithms are discussed in detail in the Chapters 3 to 6.

2.1 Recognition and Classification Pipelines

3-D recognition and classification pipelines usually consists of four steps. However, in existing approaches we often find additional preprocessing steps to ease segmentation. This allows the execution of subsequent algorithms exclusively on “interesting” parts of the scene and is done, for example, in a recognition pipeline described by Rusu et al. in [69]. In the context of this work it is assumed that the objects to be recognized are sufficiently segmented. Thus, this (preprocessing) step is not discussed.

Therefore, the pipeline starts with the selection of points to be used for the extraction of the feature descriptions. In many cases, these points are determined by sparse sampling, e. g., in the pipelines described by Johnson and Hebert [37], Frome [23], Drost [17], and Aldoma [3]. By contrast, keypoint detection algorithms are used less frequently in current pipelines, such as in those described by Chen and Bhanu [10], Zhong [105], and Mian [56].

Usually, the next step is the determination of local 3-D feature descriptions on all selected points with methods described in Section 4. When all local features have been determined, the next steps depend on the purpose of the pipeline. If the pipeline is used for recognition tasks, objects are often searched within a point cloud which represents a whole scene, but not a single object. In this case, the next step is called hypothesis generation which considers all objects that could be found in the scene.

Zhong, Drost et al. and Mian et al. use a method called *pose clustering* [105, 17, 56]. When performing pose clustering, each local 3-D feature description of the scene is compared to the features of the searched objects. Each pair of matching features is labeled with the corresponding object ID. Sets of size k of feature pairs with the same label are used to compute transformations between the object and the corresponding features in the scene. Finally, a clustering of these transformations is performed to find the transformation, which is supported by the most feature pairs.

Another technique used by Aldoma et al. [3] is called *geometric consistency*. They compute all pairs of features as shown above and select one of the pairs randomly as an initial element of a set $\mathcal{G}$. Then they start adding all feature pairs to $\mathcal{G}$, which are geometrically consistent to all pairs already within $\mathcal{G}$ until no more consistent feature pairs can be found. This is repeated several times



Figure 2.2: The generic steps of a recognition pipeline.

with different initial elements. Finally, the largest set $\mathcal{G}$ is used as a hypothesis, since this set corresponds with a high probability to the desired object.

In both approaches a final verification of the hypothesis is performed by comparing the individual features of the searched objects and the scenes. Therefore, a generic recognition pipeline can be summed up as shown in Figure 2.2.

In contrast to a recognition pipeline, where a specific instance of an object is searched within a scene, a classification pipeline requires other, more abstract steps subsequently to the determination of the feature descriptions. A common way to classify objects based on local descriptions is a combination of a *bag of words* or *bag of features* model, respectively, and a classifier [90, 91, 41, 52, 51, 75, 103]. The bag of features model is used to quantize the large number of possible variants of high-dimensional feature descriptions to a finite vocabulary. The creation of vocabulary in a context of a bag of features model is often done in a preprocessing step with a k -means algorithm [90, 91, 41, 52, 51, 103]. Furthermore, Jégou et al. describe a way to improve the quality of the vocabulary words [36]. During the classification each of the feature descriptions is mapped to its corresponding word and filled into a histogram with the same number of bins as the size of the vocabulary. This histogram is called *frequency histogram*. That way a global description of the object emerges. Frequency histograms are used as input vectors for classifiers. All of the pipelines except the approach proposed by Yi use support vector machines (SVM) as classifiers. Yi et al. [103] perform a comparison of their own approach in which they use a language model with approaches using SVMs and conditional random fields (CRF).



Figure 2.3: A generic representation of a classification pipeline.

Therefore, under the assumption of already segmented point clouds, a classification pipeline can be summarized with the four main steps shown in Figure 2.3.

2.2 Keypoint Detection

3-D data processing algorithms are computationally expensive. Accordingly, it is useful to reduce the set of points to a much smaller subset. Depending on the input data, a simple way to achieve this is to calculate a sparse sampling or a mesh decimation. In addition, a comparison of uniform and recurrent regions of point clouds is hardly helpful. Hence, it makes sense to identify regions of interest, i. e., qualified keypoints in terms of repeatability and informativeness, and the calculation of keypoints should be, whenever possible, the first step towards a local feature based 3-D classification system.

A survey from Guo et al. [27] from the year 2014 provides a comprehensive overview of the available keypoint detection algorithms. In addition, two recent evaluations can be found in the works of Bronstein et al. [8] and Salti et al. [72].

Although the keypoint detection algorithms are often developed in conjunction with feature description algorithms, in some cases there are individual keypoint detection algorithms or feature description algorithms without a specific keypoint algorithm. Therefore, the methods are – also with regard to the pipeline – treated separately. A detailed description of individual algorithms can be found in Chapter 3 of this thesis.

2.3 3-D Object Description

To compare and match 3-D point clouds, an object description, which should be as unique as possible, is necessary. This description of a 3-D object is a major challenge, especially when noise and occlusion should be taken into account. Descriptors for 3-D objects can be roughly divided into three categories:

- local 3-D feature descriptor
- global 3-D object descriptor
- hybrid solutions of the above mentioned approaches, where local 3-D feature descriptors are combined into a global descriptor algorithm

Common to all these methods is, that they generate n -dimensional signatures or histograms with n bins, which can be used for classification or recognition tasks.

In both cases, the term feature vector is often used synonymously to local feature descriptions.

Additionally, there is a kind of generic global descriptor, which can be used in many different environments. This aforementioned generic global descriptor, namely the bag of words model, is a histogram containing the occurrences of each “word” of a precomputed “vocabulary”. In context of local 3-D feature descriptions, the vocabulary is determined by a clustering of feature descriptions, where each element of the vocabulary is a centroid of a cluster. Therefore, this method is also called “bag of features”. Further details regarding the bag of features approach are located in Section 4.3.

2.3.1 Local 3-D Feature Descriptors

A local 3-D feature description is a representation of the local neighborhood at a certain 3-D point, which is typically given by a keypoint. At this point, geometric information of the local surface, i. e., the local 3-D point cloud around that keypoint can be extracted and encoded into a feature description. The existing methods can be divided roughly into three broad categories: signature based, histogram based, and transform based methods.

Most of the currently available local 3-D feature description algorithms can be found in the survey from Guo et al. [27]. Since these algorithms are an essential part of this thesis, a detailed description of the methods can be found in Section 4.1.

In addition, there are some evaluations of the proposed algorithms: Bustos et al. [9], Heider et al. [30], and Alexandre [4].

2.3.2 Global 3-D Descriptors

The goal of global descriptors is to process a 3-D object as a whole for recognition, regardless of it being a mesh or a point cloud. One of the simplest options to describe a 3-D object is a bounding box aligned along the principle axes. This reduces the object to three absolute lengths or two values that describe the length ratios of the principal axes. The significance of these values is indeed very limited.

However, many methods have been developed in the 90s and the beginning 21st century, which allow a more detailed description of 3-D objects. Based on the

article by Bustos et al. [9], these global description algorithms can be categorized with respect to the underlying approaches used: descriptions with conceptually different types of shape information, statistical approaches, and structural 3-D object shape description that can be represented in the form of a graph.

Although no global 3-D descriptors will be used in the context of this work, for the sake of completeness some selected methods will be mentioned hereinafter:

Suzuki et al. [85] introduced a rotation invariant shape descriptor which uses the point cloud data only. They fit the point cloud into a unit cube, divide the cube into a coarse grid and count the points in each grid cell.

Vranić and Saupe [98] and Lucchese et al. [50] proposed FFT based shape descriptors for voxelized representations of point clouds. They divide a 3-D point cloud into a voxel grid and use it as input for a 3-D Fourier transform. While Vranić and Saupe use the absolute values of the obtained coefficients as description, Lucchese et al. use the slice theorem to calculate radial projections, which they compare.

Other methods use the voxelized volume data of normalized models to make a direct comparison. Approaches of this kind are used in the shape based similarity search from Keim [40] and a system from Paquet et al. [58, 59] called Nefertiti.

The method for similarity search on 3-D databases by Heczko et al. [29] creates a set of descriptions. The first description consists of length values of a set of rays beginning at the mass center of the 3-D objects and ending at the surface of the object. The second description consists of volume values of 6 sliced pyramids with its peak at the mass center and its floors forming cube around the object. Furthermore, Heczko et al. use a parallel projection onto the 6 faces of a bounding cube aligned along the principal axes of the 3-D object. Then they apply a Fourier transform on the resulting silhouettes and use the absolute Fourier coefficients as global description. In a second variant Heczko et al. create normalized depth images instead of silhouettes on the 6 faces of the bounding cube.

Saupe et al. [74] and Vranić et al. [99] use spherical harmonics to approximate the objects shape. The parameters of the spherical harmonics will be used as description.

Furthermore, there are many other publications on the subject of global 3-D object descriptions and the list is by no means complete. However, many of them are based on one of the methods mentioned above.

2.3.3 Hybrid Feature Descriptors

The following methods and algorithms can be understood as a combination of local features to build a single global description for larger regions or full objects in 3-D point clouds, and will hereinafter be referred to as *hybrid* feature descriptors.

Such methods essentially summarize the local feature description to a single global description in a subsequent step. While this principle, depending on the objective, is also typically used within classification pipelines (see Section 2.1), hybrid solutions enable the simplified use of the local feature descriptions (on which they are based).

Since within this thesis local 3-D feature descriptions are always combined with the already mentioned bag of features approach, these methods are not used in context of this thesis. Nevertheless, for completeness some of them are briefly described:

One early hybrid method is the local feature histogram introduced by Hetzel et al. [31] in 2001. The work is intended to be used on depth images. Hetzel et al. mention three characteristics which can be obtained for each individual point of the depth image: the distance between neighboring points, the surface normals described by φ and θ , and the shape index S_i which is explained in the section on local surface patches (Section 3.1.4). By combining these three features, i.e., four values for all points of interest in a single 4-D histogram, Hetzel et al. use this histogram as global description for an 3-D object. They performed several tests to assess the number of bins with the highest recognition rate. The best recognition results were achieved with a combination of all three features and with 8 bins for depth values, 4 bins for both, φ and θ , and with 8 bins for the shape index.

Drost et al. [17] introduced a hybrid feature descriptor which they called point pair feature. Point pair features are rotationally invariant without the need of a local reference frame, but require the approximation of normal vectors for each point of a point cloud or a mesh. Point pair features are 4-tuples, that describe the relative orientation and distance between two 3-D points $\mathbf{p}_i$ and $\mathbf{p}_j$. Let $\mathbf{d} = \mathbf{p}_i - \mathbf{p}_j$ be the displacement vector between those two points. Then the feature consists of the Euclidean length of $\mathbf{d}$, the angle between the normal vector

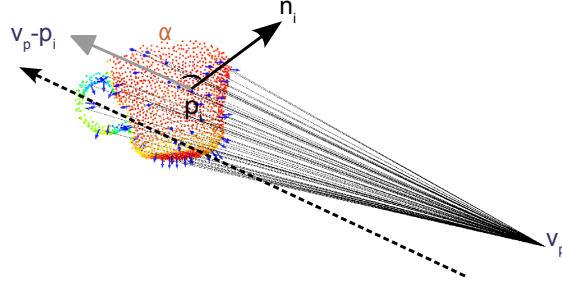


Figure 2.4: The computation of the viewpoint component of the viewpoint feature histogram from [67].

$\mathbf{n}_i$ and the displacement vector $\mathbf{d}$, the angle between the normal vector $\mathbf{n}_j$ and $\mathbf{d}$, and the angle between the two normals:

$$(2.1) \quad f(\mathbf{p}_i, \mathbf{p}_j) = \left(\|\mathbf{d}\|_2, \cos^{-1}(\mathbf{n}_1 \cdot \mathbf{d}), \cos^{-1}(\mathbf{n}_2 \cdot \mathbf{d}), \cos^{-1}(\mathbf{n}_1 \cdot \mathbf{n}_2) \right)$$

Since this method was developed specifically for object recognition, Drost et al. compute a global model representation based on these local feature. This global representation is a hash table where the keys are quantized and mapped representations of the local features and the values are sets of point pairs with similar local features. On the one hand this allows a very efficient mapping between local features of an input scene and those of the model, and on the other hand a comparison of the spatial dependencies between all local features that are associated with the key of the hash table.

The viewpoint feature histogram is part of a full recognition pipeline for robots introduced by Rusu et al. [67] in 2010 and is a hybrid descriptor for point clouds. A larger part of this descriptor has been taken over from the fast point feature histogram introduced in Section 4.1.6. The viewpoint feature histogram is a concatenated histogram filled with angles from different sources. For the first component they determine a direction $\mathbf{d} = \mathbf{v}_p - \mathbf{p}_i$, where $\mathbf{v}_p$ is the viewpoint and $\mathbf{p}_i$ is the central point of a 3-D point cloud as depicted in Figure 2.4. The angles between this central viewpoint direction and each of the normal vectors, i.e., $\cos^{-1}(\mathbf{d} \cdot \mathbf{n}_j)$, are filled into the histogram.

The second component consists of the three angles α , φ , and θ , determined with the method which has been described in the context of the fast point feature histogram, relative to the central viewpoint direction $\mathbf{d}$. Figure 2.5 depicts an example of a viewpoint feature histogram.

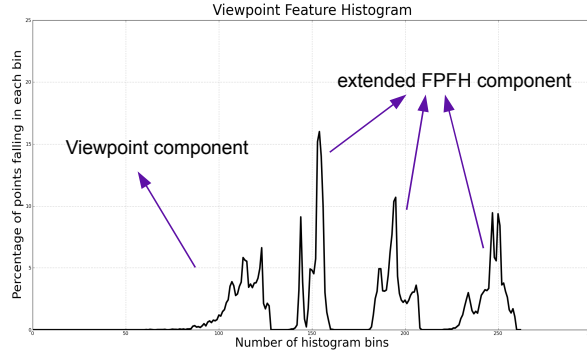


Figure 2.5: An example of a viewpoint feature histogram obtained from [67].

2.4 Reinforcement Learning

Machine learning and artificial intelligence were originally motivated by biological nervous systems. The aim was to construct the algorithms in a way, that enables a computer system to mimic or simulate the human intelligence. Today, machine learning is used in a variety of tasks and typically classified into three broad categories [64].

If the learning algorithm is left on its own to find structure in its input, e. g., for clustering or pattern recognition tasks, the algorithm does unsupervised learning. Another class of learning algorithms can use a “teacher” which describes the coherences of a system by exemplary observations and creates training data, by means of which a machine learning system can derive the phenomenological relationships and learn. This type of algorithm is called supervised learning. The third class of machine learning algorithms is reinforcement learning (RL) [84, 83]. The main characteristics of reinforcement learning are reflected by the following quotation from the book by Richard S. Sutton and Andrew G. Barto:

“The idea that we learn by interacting with our environment is probably the first to occur to us when we think about the nature of learning. When an infant plays, waves its arms, or looks about, it has no explicit teacher, but it does have a direct sensorimotor connection to its environment. Exercising this connection produces a wealth of information about cause and effect, about the consequences of actions, and about what to do in order to achieve goals.

Throughout our lives, such interactions are undoubtedly a major source of knowledge about our environment and ourselves. Whether we are learning to drive a car or to hold a conversation, we are acutely aware of how our environment responds to what we do, and we seek to influence what happens through our behavior. Learning from interaction is a foundational idea underlying nearly all theories of learning and intelligence.” (Sutton and Barto [83]).

A quick overview on reinforcement learning can be found in a book of Szepesvári [87], and large a collection of surveys about methods from different fields of reinforcement learning can be found in a book of Wiering and Otterlo [101]. Reinforcement learning in general, and in particular the method which will be used in this thesis named Q -learning by Watkins [100] will be introduced and discussed in Chapter 6.

A preliminary concept of the reinforcement learning based approach introduced in this work was presented by Garstka and Peters [24].

Keypoints, also referred to as interest points, are points in an image or 3-D point cloud that should be stable and distinctively describe an interesting region of a point cloud. Typically, the number of interest points in a 3-D point cloud is only a small subset of all 3-D points. The goal of keypoint detection algorithms is to support the local feature descriptors, and thus a keypoint detection algorithm is often developed in combination with a local feature descriptor.

Subsequently, the keypoint detection algorithms are presented in chronological order and classified according to important characteristics. According to Guo et al. [27], these characteristics are:

1. The *type of data* on which the algorithm is applied to, distinguishing between meshes, point clouds and depth images.
2. The *scalability* of the keypoint detection method is divided into fixed scale and adaptive scale methods. While fixed scale methods use a predetermined parameter, which describes the radius of the local region to be used for the determination of keypoints, adaptive methods build a scale-space to pick distinctive keypoints at different scales.
3. The *method* maps each algorithm to a basic process concept. For fixed scale approaches, a distinction is made between curvature based and other methods. Adaptive approaches are differentiated into coordinate smoothing, geometric attribute smoothing, and surface variation based methods.

Since the focus of this work lies on unstructured point data, only algorithms for point clouds and depth images are discussed in this section. An overview of the available keypoint detection methods for meshes can be found in the aforementioned survey from Guo et al. [27].

3.1 Keypoint Detection Algorithms

3.1.1 Multi-scale Feature Extraction on Point-Sampled Surfaces

The keypoint detection algorithm introduced by Pauly et al. [60] in 2003 is an *adaptive* method designed for *point clouds* and determines the keypoint due to the calculation of *surface variations*.

Given an unstructured point cloud $\mathcal{P}$, the set of the n nearest neighbors of a point $\mathbf{p} \in \mathcal{P}$ is $\mathcal{N}_{\mathbf{p},n}$. Let $\bar{\mathbf{p}}$ the centroid of $\mathcal{N}_{\mathbf{p},n}$ and $\hat{\mathbf{p}}_i = \mathbf{p}_i - \bar{\mathbf{p}}$ with $\mathbf{p}_i \in \mathcal{N}_{\mathbf{p},n}$ the difference vector between the centroid and the i -th nearest neighbor. The covariance matrix $\mathbf{C}$ of $\mathcal{N}_{\mathbf{p},n}$ is defined as

$$(3.1) \quad \mathbf{C} = \frac{1}{k} \begin{bmatrix} \hat{p}_{1,x} & \hat{p}_{1,y} & \hat{p}_{1,z} \\ \vdots & \vdots & \vdots \\ \hat{p}_{n,x} & \hat{p}_{n,y} & \hat{p}_{n,z} \end{bmatrix}^\top \cdot \begin{bmatrix} \hat{p}_{1,x} & \hat{p}_{1,y} & \hat{p}_{1,z} \\ \vdots & \vdots & \vdots \\ \hat{p}_{n,x} & \hat{p}_{n,y} & \hat{p}_{n,z} \end{bmatrix}.$$

Then the surface variation $\sigma_n(\mathbf{p})$ introduced by Pauly et al. is

$$(3.2) \quad \sigma_n(\mathbf{p}) = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2},$$

where λ_0 , λ_1 , and λ_2 are the eigenvalues of $\mathbf{C}$ with $\lambda_0 \leq \lambda_1 \leq \lambda_2$.

If the surface variation $\sigma_n(\mathbf{p})$ is used as weight ω for each point, then weights for different scales can be obtained by varying the neighborhood size n . The covariance matrix, as proposed by Pauly et al., is defined as sums of squared distances from the neighborhoods centroid. If the neighborhood size n is increased, high-frequency structures are attenuated. For this reason, the neighborhood size n can be used as discrete scale parameter.

On the one hand a user can select a proper scale for their application itself. On the other hand Pauly et al. propose an automatic scale selection based on a method introduced by Lindeberg [46] and used, for instance, by the well known image feature extraction SIFT by Lowe [49]. The keypoints are obtained from the points at which the weights attain extrema with respect to the scale level. In comparison to the 2-D approaches for images, the spatial coordinates will be ignored, as they are unstructured.

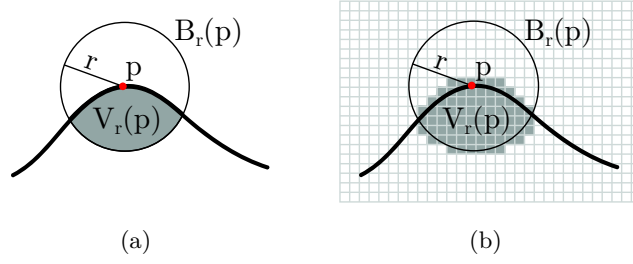


Figure 3.1: These two figures from [26] depict the integral descriptor for a point $\mathbf{p}$ of a curvature. The value of the descriptor is the proportion of the interior V_r of a circle B_r with radius r intersected by the curvature. An approximation of the continuous case (a) can be computed efficiently using a grid (b).

In order not to use a single weight for the determination of a keypoint, and to avoid problems due to noise, Pauly et al. recommend to count the number of times a surface variation exceeds a threshold $\sigma_{\max}$. For this, they define

$$(3.3) \quad \Omega(\mathbf{p}, n) = \begin{cases} 1 & \sigma_n(\mathbf{p}) > \sigma_{\max} \\ 0 & \sigma_n(\mathbf{p}) \leq \sigma_{\max} \end{cases}$$

and use

$$(3.4) \quad \omega(\mathbf{p}) = \sum_n \Omega(\mathbf{p}, n)$$

as persistent point weight, which can be used to identify persistent keypoints over different scales.

3.1.2 Integral Volume Descriptor

The keypoint detection algorithm introduced by Gelfand et al. [26] in 2005 is an *adaptive* method, that can be used with *meshes*. It determines the keypoint due to the calculation of *surface variations*. The approach is based on the method of Manay et al. [53] and adopts the concept of integral invariant signatures from 2-D to 3-D.

Figure 3.1 depicts the concept of this descriptor in 2-D. For each point of a curvature, the surface ratio of a local area, which is intersected by the curvature, is calculated (Figure 3.1 (a)). By the discretization of the local area (commonly a circle), an approximation of the ratio can be calculated efficiently (Figure 3.1 (b)).

As already mentioned, Gelfand et al. extended the 2-D integral descriptor to 3-D. While the integration kernel $B_r(\mathbf{p})$ is a sphere of radius r centered at the

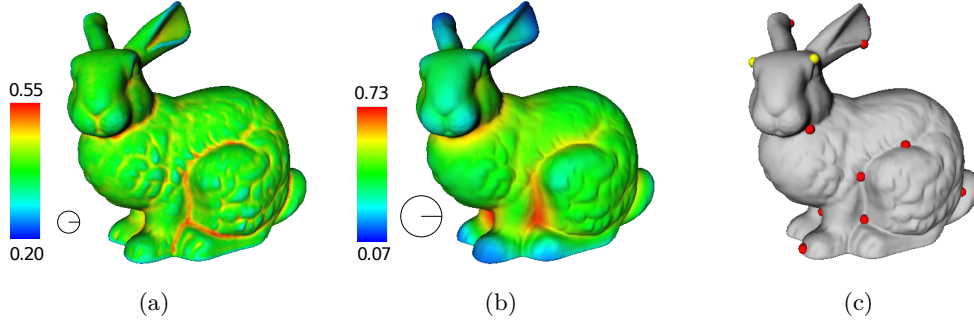


Figure 3.2: These figures from [26] show the values of normalized $V_r(\mathbf{p})$ of the 'Bunny' (The Stanford 3-D Scanning Repository [13]). The figures (a) and (b) show the values for a smaller and a larger convolution sphere. Figure (c) shows the keypoints identified by this algorithm. The yellow markers show stable keypoints for (a), while the red markers show corresponding features for (b).

point $\mathbf{p}$, the value $V_r(\mathbf{p})$ is the interior part of the sphere intersected by the input mesh. Analogous to the method of Manay et al. the discretization of the local area is implemented using a voxel grid. This enables a fast approximation of $V_r(\mathbf{p})$ convolving a voxelized input object with $B_r(\mathbf{p})$. The convolution results, i. e., the approximations of $V_r(\mathbf{p}) \forall \mathbf{p} \in \mathcal{P}$, are filled into a histogram with a bin size according to Scott's rule. 3-D points with rare $V_r(\mathbf{p})$ values, i. e., 3-D points of less filled histogram bins, will be used as keypoint candidates.

Figure 3.2 shows an example result based on the method of Gelfand et al. To be able to use the same method for depth images and point clouds, the algorithm needs to be extended. This extension by Garstka et al. [25] primarily consists of the following additional steps:

1. The estimation of the mean distance between 3-D points (hereinafter referred to as point cloud resolution) to get an appropriate voxel size.
2. The determination of watertight voxel representations for point clouds and depth images.

In a mesh the mean distance between 3-D points is defined by Johnson et al. [37] as median distance between vertices in the mesh. To find appropriate parameters for an approximation of the point cloud resolution in terms of a corresponding mesh resolution, experiments indicate, that the mean distance of the 7 nearest neighbors is the best choice to approximate this value. The point

cloud resolution is used as voxel size for a cubic voxel grid. Each voxel containing a 3-D point is initialized with a value of 1. In the case that the 3-D data is a depth image, the 3-D points will be calculated by back-projection and the x - and y -axis of voxel grid will be aligned to the x - and y -axis of the depth image.

Both, depth images and point clouds require an approximation of the surface to create a watertight model. In 3-D point clouds based on depth images, related 3-D points can be determined easily based on the corresponding depth pixels. The watertight voxel grid will be computed filling all voxels along the z -axis in ascending order beginning with the first voxel intersected by the approximated surface.

For pure 3-D point clouds a closed model is assumed. One possible approach to create a watertight model is the method by Adamson et al. [1]. They use spheres around all points of the point cloud to dilate the points to a closed surface. Another approach by Hornung et al. [33] is based on a combination of adaptive voxel scales and local triangulations. The major drawback of both methods consists in their long computation times.

For this reason, Garstka et al. have developed a fast and highly parallelizable approach, with the minor disadvantage, that the algorithm may provide inaccurate results in a few cases. Each voxel containing a 3-D point is initialized with 1. All other voxel values are set to 0. For each dimension x , y and z the algorithm iterates over the voxels. Beginning with the first occurrence of a voxel with a value of 1, the following ones will be decreased by 1 until the next voxel of value 1 is reached. This steps will be repeated until the boundary of the voxel grid is reached. If the boundary is reached while decreasing the voxel values by 1, there is at least one hole in the surface. In this case all values of this iteration step will be set back to previous values. Finally, all voxels with a value ≤ -2 , i. e., which have been marked as inner voxels by at least two passes, will be treated as inner voxels. These voxels will get a value of 1, while all other voxels will get a value of 0. In a post-processing step all holes, i. e., tubes of size 1 will be filled. Both scenarios are shown in Figure 3.3.

At this point a convolution of the voxel grids with a spherical kernel, as described by the work of Gelfand et al., is possible.

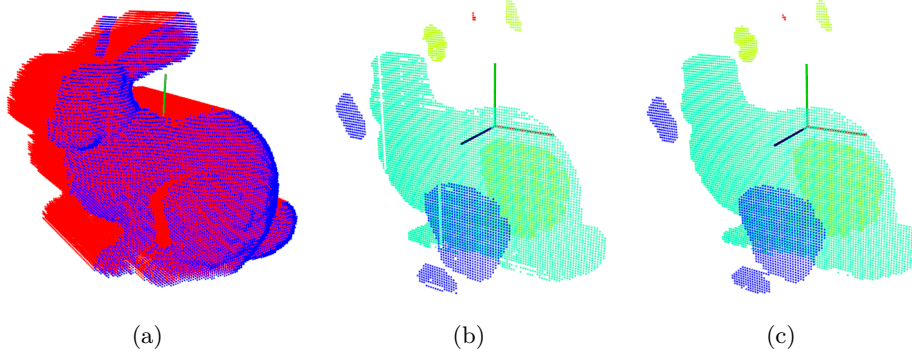


Figure 3.3: Extensions for the integral volume descriptor. Figure (a) shows a voxelized depth image from the 'Stanford Bunny'. Blue dots represent voxels with corresponding 3-D points. Red dots represent the filled voxels. The figures (b) and (c) show sliced versions of the filled voxel grid of pure point cloud from the 'Stanford Bunny'. In (b) the tubes that occur if voxelized surface contains a hole, are visible. Figure (c) shows the result after the post-processing step where those tubes got filled.

3.1.3 Selection by the Smallest Eigenvalues of Local Regions

The keypoint selection used by Matei et al. [54] in 2006 is a *fixed scale* method, that can be used with *point clouds*. It determines the keypoint due to the scatter of a local region measured with the smallest eigenvalue of the local region.

The approach is relatively easy and straightforward. For each 3-D point $\mathbf{p} \in \mathcal{P}$, Matei et al. compute a normalized covariance matrix

$$(3.5) \quad \mathbf{C}(\mathbf{p}) = \frac{1}{n_{\mathbf{p}}} \sum_i (\hat{\mathbf{p}}_i - \mathbf{p}) \cdot (\hat{\mathbf{p}}_i - \mathbf{p})^\top,$$

which they call a scatter matrix, where $n_{\mathbf{p}}$ is the number of points $\hat{\mathbf{p}}_i$ with a distance $\|\hat{\mathbf{p}}_i - \mathbf{p}\|$ less than w .

From the eigenvalues for each $\mathbf{C}(\mathbf{p})$ with $\lambda_0^{\mathbf{C}(\mathbf{p})} \geq \lambda_1^{\mathbf{C}(\mathbf{p})} \geq \lambda_2^{\mathbf{C}(\mathbf{p})}$ the smallest eigenvalues $\lambda_2^{\mathbf{C}(\mathbf{p})}$ will be sorted into a list. Finally, the selection of the keypoints is done in a greedy manner based on the sorted list: the highest eigenvalue is removed from the list and the corresponding 3-D point is used as keypoint, if the distance to already chosen keypoints is larger than d . Otherwise the 3-D point is discarded.

3.1.4 Local Surface Patches

This keypoint detection algorithm introduced by Chen and Bhanu [10] in 2007 is a *fixed scale* method, that can be used with *depth images*. The algorithm uses *curvatures* as distinctiveness measure to detect the keypoints.

To estimate the curvatures, they fit a quadratic surface $f(x, y) = ax^2 + by^2 + cxy + dx + ey + f$ to a local window. The quadratic surface is used to determine, among other things, the surface normals and the principal curvatures. On this basis Chen and Bhanu define a shape index S_i for a 3-D point $\mathbf{p}$ as follows:

$$(3.6) \quad S_i(\mathbf{p}) = \frac{1}{2} - \frac{1}{\pi} \tan^{-1} \frac{k_1(\mathbf{p}) + k_2(\mathbf{p})}{k_1(\mathbf{p}) - k_2(\mathbf{p})},$$

where k_1 and k_2 are the maximum and minimum principal curvatures, respectively. Let $\mathbf{p}$ be the center point of a local window, then $\mathcal{M}_{\mathbf{p}}$ defines the set of all points in the local window, and accordingly

$$(3.7) \quad \mu = \frac{1}{|\mathcal{M}_{\mathbf{p}}|} \sum_{\hat{\mathbf{p}} \in \mathcal{M}_{\mathbf{p}}} S_i(\hat{\mathbf{p}})$$

is the mean of shape indexes in the local window. The point $\mathbf{p}$ is marked as keypoint, if its shape index satisfies the following condition:

$$(3.8) \quad \begin{aligned} S_i(\mathbf{p}) &= \max_{\hat{\mathbf{p}} \in \mathcal{M}_{\mathbf{p}}} S_i(\hat{\mathbf{p}}) \quad \text{and} \quad S_i(\mathbf{p}) \geq (1 + \alpha) \cdot \mu \\ &\quad \text{or} \\ S_i(\mathbf{p}) &= \min_{\hat{\mathbf{p}} \in \mathcal{M}_{\mathbf{p}}} S_i(\hat{\mathbf{p}}) \quad \text{and} \quad S_i(\mathbf{p}) \leq (1 - \beta) \cdot \mu \end{aligned}$$

The parameter α and β control the keypoint selection. In summary, $\mathbf{p}$ is a keypoint if its shape index is a local optimum (minimum/maximum) within a local window, where $\mathbf{p}$ is the center point. An example is shown in Figure 3.4.

3.1.5 Thrift

The algorithm presented by Flint et al. [22] in 2007 is a 3-D extension of 2-D algorithms like SIFT and SURF called THRIFT. It is an *adaptive* algorithm for *point clouds* using the *coordinate smoothing* approach.

Flint et al. construct a density function $f(\mathbf{p})$ for a 3-D point cloud. They divide the spatial space by a uniform voxel grid $\mathcal{V} = \{V_{i,j,k}\}$, where $(i, j, k) \in I \subset$

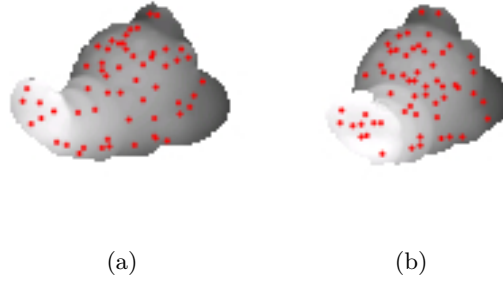


Figure 3.4: Local surface patches. The figures (a) and (b) from [10] show the keypoints identified for one object and two different views.

$\mathbb{Z}^3$ are the indexes of the voxels in each spatial dimension. With the quantity of 3-D points $|V_{i,j,k}|$ the normalized quantity of each voxel is

$$(3.9) \quad D = \frac{|V_{i,j,k}|}{\max_{(i,j,k) \in I} |V_{i,j,k}|}.$$

With the center $\mathbf{c}_{i,j,k}$ of each voxel, the density function is a sum of delta functions

$$(3.10) \quad f(\mathbf{p}) = \sum_{(i,j,k) \in I} D(i,j,k) \cdot \delta(\mathbf{p} - \mathbf{c}_{i,j,k}).$$

To construct a density scale-space Flint et al. convolve $D(i,j,k)$ with a series of 3-D Gaussian kernels

$$(3.11) \quad g(\mathbf{p}, \sigma) = \exp \left(\frac{-\mathbf{p}_x^2 - \mathbf{p}_y^2 - \mathbf{p}_z^2}{2\sigma^2} \right),$$

where the scale of each layer l is defined by $\sigma_{l+1} = m \cdot \sigma_l$. Flint et al. use a factor $m = 2$ for efficiency. This gives rise to a scale-space for $D(i,j,k)$ with

$$(3.12) \quad S(\mathbf{p}, \sigma) = (D \otimes g(\mathbf{p}, \sigma)).$$

Finally, they compute the determinant of Hessian matrix $|\text{Det}(\mathbf{H}(\mathbf{p}, \sigma))|$ at each point of the scale space. Within the resulting $3 \times 3 \times 3 \times 3$ matrix, a non maxima suppression reduces the entries to local maxima, which become interest points.

3.1.6 Multi-Scale Interest Regions from Point Clouds

The keypoint detection algorithm introduced by Unnikrishnan and Hebert [95] in 2008 is an *adaptive* method designed for *point clouds*. It determines the keypoints based on *surface variations*.

Unnikrishnan and Hebert introduce an integral operator $\tilde{A}$ for non-uniformly sampled sensor data. For this purpose they start with the continuous case where $A : \mathbb{R}^d \times \mathbb{R}^+ \rightarrow \mathbb{R}^d$ for $d = 2$ is defined as

$$(3.13) \quad A(\alpha(s), t) = \int_{\Gamma} \phi(s, u, t) \alpha(u) du,$$

where $\alpha(s) : \mathbb{R} \rightarrow \mathbb{R}^2$ is a parameterized function of a curve Γ and $\phi(s, u, t)$ is a Gaussian integral kernel

$$(3.14) \quad \phi(s, u, t) = (2\pi t^2)^{-\frac{1}{2}} \exp\left(-\frac{(s-u)^2}{2t^2}\right).$$

With the assumption, that the function is parameterized so that $\alpha(0) = \mathbf{x}$ is a point on this curve, they infer

$$(3.15) \quad A(\alpha(0), t) \approx \mathbf{x} + \kappa(\mathbf{x}) \mathbf{n}_{\mathbf{x}} \frac{t^2}{2},$$

where $\kappa(\mathbf{x})$ is the curvature at $\mathbf{x}$ and $\mathbf{n}_{\mathbf{x}}$ is the normal vector of $\mathbf{x}$. In other words, $A(\mathbf{x}, t)$ displaces $\mathbf{x}$ along its normal vector in a proportion of $\kappa(\mathbf{x})$.

In a second step Unnikrishnan and Hebert extend this approach to 3-D. Let Π_{θ} be a plane that contains $\mathbf{n}_{\mathbf{x}}$ and whose normal lies in the tangent space of $\mathbf{x}$ at an angle θ to some reference tangent. The intersection of this plane and the surface is a curve described by a parameterized function $\alpha_{\theta}(s)$ with $\alpha_{\theta}(0) = \mathbf{x}$. With the surface property, that the mean of the curvature of two orthogonal tangent directions θ and $\theta + \pi/2$ is equivalent to the mean curvature at $\mathbf{x}$, they extend equation (3.15) to

$$(3.16) \quad A(\mathbf{x}, t) \approx \mathbf{x} + \left[\kappa_{\theta}(\mathbf{x}) + \kappa_{\theta+\pi/2}(\mathbf{x}) \right] \mathbf{n}_{\mathbf{x}} \frac{t^2}{2}.$$

They adopt this approach for a non-uniform sampling by eliminating the invariance of the sampling distribution using the geodesic distance $d_{\mathbb{G}}^2(\mathbf{x}_i, \mathbf{x}_j)$ between all 3-D points. Furthermore, to create a scale-space, they define multiple scales with $t_k = t_0 1.6^k$, where t_0 is the base scale. This leads to an operator

$$(3.17) \quad \tilde{A}(\mathbf{x}_i, t) = \frac{\sum_j \tilde{\phi}(\mathbf{x}_i, \mathbf{x}_j, t) \mathbf{x}_j}{\sum_j \tilde{\phi}(\mathbf{x}_i, \mathbf{x}_j, t)},$$

where the weight for each pair of points is

$$(3.18) \quad \begin{aligned} \tilde{\phi}(\mathbf{x}_i, \mathbf{x}_j, t) &= \frac{\phi(\mathbf{x}_i, \mathbf{x}_j, t)}{p_t(\mathbf{x}_i)p_t(\mathbf{x}_j)}, \text{ with} \\ \phi(\mathbf{x}_i, \mathbf{x}_j, t) &= (2\pi t^2)^{-\frac{1}{2}} \exp\left(-\frac{d_{\mathbb{G}}^2(\mathbf{x}_i, \mathbf{x}_j)}{2t^2}\right) \text{ and } p_t(\mathbf{x}_i) = \sum_j \phi(\mathbf{x}_i, \mathbf{x}_j, t). \end{aligned}$$

Finally, they compute the invariant as

$$(3.19) \quad F(\mathbf{x}_i, t) = \frac{2\|\tilde{A}(\mathbf{x}_i, t) - \mathbf{x}_i\|}{t} \exp\left(-\frac{2\|\tilde{A}(\mathbf{x}_i, t) - \mathbf{x}_i\|}{t}\right),$$

which is an exponentially damped displacement value of the point $\mathbf{x}_i$ in a local area of size t . If $F(\mathbf{x}_i, t)$ is an extremum within the geodesic neighborhood and the scale, $\mathbf{x}_i$ is used as keypoint.

3.1.7 2.5-D SIFT

In 2009 Lo and Siebert [48] presented an *adaptive* algorithm for keypoint detection in *depth images*. Their *coordinate smoothing* method is inspired by 2-D SIFT from Lowe [49].

To extract keypoints from depth images Lo and Siebert create a smoothed version of the depth image, which is upsampled by a factor of 2. This is done to avoid alias effects and false keypoints at sharp boundaries. Furthermore, they normalize the depth values to a standard normal distribution, i. e., with the mean/expectation $\mu = 0$ and the standard deviation $\sigma = 1$ to constrain potentially large ranges.

Based on this depth image they build a difference of Gaussians (DoG) scale-space and identify extrema by comparing a pixel to its 26 neighbors in regions of size 3×3 at the current and adjacent scales in the same way Lowe does for 2D SIFT [49]. Based upon experiments Lo and Siebert define a threshold of 0.003 between the value of a keypoint pixel and its surrounding values to reject keypoints with relatively small differences between depth values.

Finally, they filter all keypoint candidates where the ratio between the two principal curvatures κ_1 and κ_2 is below 5. The last named value is determined by experiments, too. The remaining points are the final keypoints.

Additionally, they compute an orientation θ for each keypoint and use the enhanced version of Lowe's orientation assignment. Therefore, they use a histogram

with 360 bins covering all possible orientations in steps of 1° . The histogram is filled with Gaussian weighted image gradients and convolved with a 1-D symmetric Gaussian kernel with a fixed size of 17 and $\sigma = 17$. In a last step they fit a quadratic polynomial to the three largest consecutive bins for each peak within 80% of the largest peak. This allows an approximation of the orientation to sub-bin accuracy.

3.1.8 Intrinsic Shape Signature

The algorithm presented by Zhong [105] in 2009 determines the keypoints on point clouds due to the calculation of *surface variations*. The size of the local regions is *fixed*.

Zhong computes the keypoints in a similar way to Pauly et al. (see Section 3.1.1) and Matei et al. (see Section 3.1.3). Given an unstructured point cloud $\mathcal{P}$, Zhong computes a covariance matrix $\mathbf{C}_{\mathbf{p}_i}$ for a point $\mathbf{p}_i \in \mathcal{P}$ and all its neighbors $\mathbf{p}_j \in \mathcal{P}$ in a spherical neighborhood of radius r :

$$(3.20) \quad \mathbf{C}_{\mathbf{p}_i} = \frac{\sum_{\|\mathbf{p}_j - \mathbf{p}_i\| < r} w_j (\mathbf{p}_j - \mathbf{p}_i)(\mathbf{p}_j - \mathbf{p}_i)^\top}{\sum_{\|\mathbf{p}_j - \mathbf{p}_i\| < r} w_j},$$

where the weight $w_i = 1/|\{\mathbf{p}_j : \|\mathbf{p}_j - \mathbf{p}_i\| < r\}|$ is the inverse of the amount of points in a spherical neighborhood of radius r of a point $\mathbf{p}_i$. This weight is used to compensate unequal distributions of points within a point cloud. For all $\mathbf{C}_{\mathbf{p}_i}$ Zhong computes the eigenvalues λ_0 , λ_1 , and λ_2 with $\lambda_0 \leq \lambda_1 \leq \lambda_2$ and introduces two constraints $\lambda_0/\lambda_1 < \tau_{0,1}$ and $\lambda_1/\lambda_2 < \tau_{1,2}$ to exclude all, in terms of local symmetries ambiguous points.

The final keypoints are selected successively from the remaining keypoint candidates. Those candidates, which are too close to already selected keypoints are skipped.

3.1.9 A Keypoint Quality Measure using Surface Variations

2010 Mian et al. [56] presented an *adaptive* keypoint detection approach, that can be used for *point clouds*, *depth images*, and *meshes*. This is a *surface variation* based method, too.

If the input is a mesh, Mian computes the normal vector $\mathbf{n}_i$ for each vertex $\mathbf{v}_i \in \mathcal{M}$. Let $\mathcal{M}_i \subset \mathcal{M}$ be the partition of the mesh, where $\|v_i - v_j\| < r, \forall \mathbf{v}_j \in \mathcal{M}_i$ for a given radius r . Now, the partition $\mathcal{M}_i$ is rotated, that the normal vector $\mathbf{n}_i$ is aligned with the positive z -axis of a local reference frame. The vertices in $\mathcal{M}_i$ are used to calculate a covariance matrix, and on this the principal axis using a principal component analysis.

Let $\hat{\mathcal{M}}_i$ be the aligned partition of the mesh. Then Mian et al. compute a ratio δ of the extrema of the x and y components of the vertices:

$$(3.21) \quad \delta = \frac{\max_x(\hat{\mathcal{M}}_i) - \min_x(\hat{\mathcal{M}}_i)}{\max_y(\hat{\mathcal{M}}_i) - \min_y(\hat{\mathcal{M}}_i)}$$

With this δ they select a vertex as keypoint, if the first principal axis is 6% longer compared to the second one, i. e., $\delta > 1.06$. This value was experimentally determined.

To use this approach with depth images and point clouds, they make a few changes. For depth images they formulate two boundary conditions to avoid key-points along boundaries. On the one hand there is a boundary of the view in a depth image. On the other hand there are boundaries caused by the self occlusion of the acquired object, which are indicated by abruptly changing depth values. In addition, they consider holes in depth images interpolating the values.

In unstructured point clouds Mian et al. have to deal with boundaries, too. Surfaces are usually non-uniformly sampled. Therefore, they introduce a soft threshold between the global density of all points and the density of the local part within radius r .

To be scale invariant, they compare multiple δ 's for different radii r . They use the scale r where the surface variation δ reaches a local maximum.

3.1.10 Scale-Space Surface Analysis in Depth Images

The work of Stückler and Behnke [82] from 2011 is an *adaptive* method designed for *depth images*. The method is based on *surface variations* and is an extension of the work of Unnikrishnan and Hebert (see Section 3.1.6). The algorithms are identical, except that Stückler and Behnke use the Euclidean distance on depth images, while Unnikrishnan and Hebert use a geodesic distance on a graph on top of a point cloud.

3.1.11 Point Feature Extraction on 3D Range Scans

The keypoint detection method proposed by Steder et al. [79, 80] in 2010 is a *fixed* keypoint detection approach with focus on *depth images*. The algorithm is named 'normal aligned radially feature', briefly NARF.

The method consists of two steps. In a first step Steder et al. label all pixels of a depth image with four scores which reflect the probability that a pixel lies on a border to the left, right, bottom and/or top. For this purpose they define a distance δ of pixels which they assume to be on the same surface as a pixel $\mathbf{p}$. The value of δ is determined for a squared neighborhood with the size of 5×5 , where $\mathbf{p}$ is the pixel at the center. They assume, that at least $m = 9$ elements lie on the same surface, which would be the case if $\mathbf{p}$ is the tip of a right angle corner. Therefore, they sort all distances between $\mathbf{p}$ and all depth values in the neighborhood in an ascending order $\{d_1 \leq \dots \leq d_{25}\}$ and select $\delta = d_9$. Then they take a look at the values in each direction (hereinafter exemplarily for the right side).

Let $\mathbf{p}_{x,y}$ be the 3-D representation of the pixel $\mathbf{p}$ lying at (x, y) . Then

$$(3.22) \quad \bar{\mathbf{p}}_{\text{right}} = \frac{1}{3} \sum_{i=1}^3 \mathbf{p}_{x+i,y}$$

is the average 3-D position of the three pixels right of $\mathbf{p}_{x,y}$. Then they calculate the distance $d_{\text{right}} = \|\mathbf{p}_{x,y} - \bar{\mathbf{p}}_{\text{right}}\|$. The final score is based on the quotient between δ and d_{right}

$$(3.23) \quad s_{\text{right}} = \max \left(0, 1 - \frac{\delta}{d_{\text{right}}} \right).$$

Since many depth sensors deliver information about unrecognized depth values or depth values out of range, Steder et al. map these values to a score $s_{\text{right}} = 1$. In addition, they check if – with respect to the depth value – $\mathbf{p}$ is in front of or behind the border. The first case indicates an obstacle border, the second a shadow border. For all obstacle borders they search in a distance of 3 pixels for a shadow border. From a potential shadow border they select that pixel with the highest score s_{shadow} and decrease s_{right} according to

$$(3.24) \quad s'_{\text{right}} = \max \left(0.9, 1 - (1 - s_{\text{shadow}})^3 \right) \cdot s_{\text{right}}.$$

Finally, if s'_{right} is above 0.8 and is a maximum regarding $\mathbf{p}_{x+1,y}$ and $\mathbf{p}_{x-1,y}$, it will be marked as obstacle border and the pixel corresponding to s_{shadow} as shadow

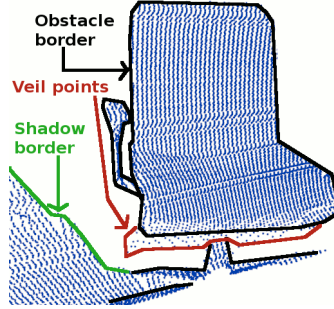


Figure 3.5: This figure from [80] illustrates different border types marked during the first keypoint detection phase.

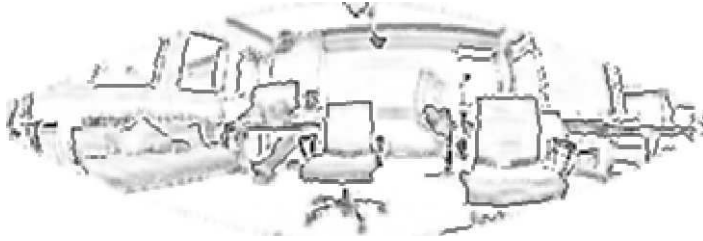


Figure 3.6: This figure from [80] shows the weights along the detected borders.

border. All pixels in between will be marked as veil points. A brief overview is depicted in Figure 3.5.

To extract keypoints from the previously computed borders, Steder et al. calculate the main direction v and the magnitude, i. e., the largest eigenvalue λ using a PCA for each pixel and its 5×5 neighborhood. In addition, each pixel gets a weight w , that is 1 for each border pixel and $1 - (1 - \lambda)^3$ for all other pixels. Figure 3.6 shows the weights of an exemplary scene.

Subsequently they determine the directions of the borders. For each 3-D point $\mathbf{p}_i$ in a neighborhood $\mathcal{N}_{\mathbf{p}_i}$ of size σ they select all points $\mathbf{p}_j$ that do not have another border in between $\mathbf{p}_i$ and $\mathbf{p}_j$ and calculate the angle of the dominant direction α_i on the plane perpendicular to the direction to the sensor. Since these directions are ambiguous, they map them into an interval of $[-90^\circ, 90^\circ]$. Finally they compute how much the surface points differ:

$$(3.25) \quad i_1(\mathbf{p}_i) = \min_{\mathbf{p}_j \in \mathcal{N}_{\mathbf{p}_i}} \left(1 - w_j \cdot \max \left(1 - \frac{10 \cdot \|\mathbf{p}_i - \mathbf{p}_j\|}{\sigma}, 0 \right) \right)$$

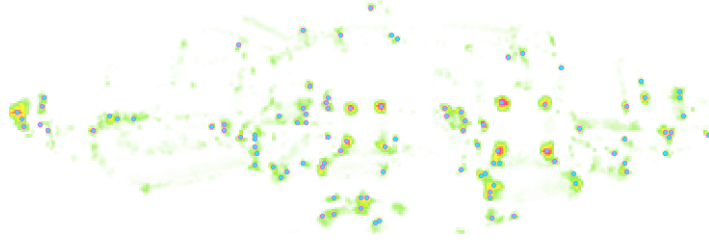


Figure 3.7: This figure from [80] shows selected keypoints corresponding to Figure 3.6.

and how much the directions differ:

$$(3.26) \quad \begin{aligned} f(\mathbf{p}_m, \mathbf{p}_n) &= \sqrt{w_n \left(1 - \left| \frac{2 \cdot \|\mathbf{p}_m - \mathbf{p}_n\|}{\sigma} - \frac{1}{2} \right| \right)}, \\ i_2(\mathbf{p}_i) &= \max_{\mathbf{p}_j, \mathbf{p}_k \in \mathcal{N}_{\mathbf{p}_i}} (f(\mathbf{p}_i, \mathbf{p}_j) f(\mathbf{p}_i, \mathbf{p}_k) (1 - |\cos(\alpha_j - \alpha_k)|)). \end{aligned}$$

Using these two values gives a score for keypoint candidates: $i(\mathbf{p}_i) = i_1(\mathbf{p}_i) \cdot i_2(\mathbf{p}_i)$. A non-maximum suppression with a static threshold selects all points that will be used as keypoints. Figure 3.7 shows the determined keypoints.

3.1.12 Interest Points of Local Surface Entropy

Fiolka et al. [20] proposed a keypoint detection algorithm in 2012. This *fixed* keypoint detection algorithm works with *point clouds* and selects the keypoints based on the distribution of local surface normals.

They approximate a surface normal $\mathbf{n}_i$ for a 3-D point $\mathbf{p}_i \in \mathcal{P}$ based on points of the local neighborhood $\mathcal{N}_{\mathbf{p}_i, r}$ with radius r . The eigenvector corresponding to the smallest eigenvalue of the covariance matrix for $\mathcal{N}_{\mathbf{p}_i, r}$ is used as a normal vector.

In order to extract regions of interest, Fiolka et al. use an entropy h with

$$(3.27) \quad h(X_{\mathcal{E}}) = - \sum_{x \in X_{\mathcal{E}}} p(x) \log p(x),$$

where $\mathcal{E} \in \mathbb{R}^3$ is a given region where the entropy is calculated for, and X is a probability distribution with the discrete probabilities $p(x)$.

To compute this entropy for a point cloud, Fiolka et al. fill all normal vectors of a region into an orientation histogram. To create an orientation histogram where each bin corresponds to an approximately equal part of the surface of a unit sphere, a uniform sampling of the sphere would be helpful. The exact

solution for uniform sampling of the sphere with equal area cells has no closed-form solution. But Fiolka et al. use an approximation of the uniform sampling introduced in [61] by Tahir Rabbani.

To get this approximation they sample the polar angle θ uniformly and change the sampling along the azimuth angle φ adaptively. Let the number of samples in θ be n_θ , then for a given value of θ_i the number of samples in φ , i. e., $n_{\varphi,i}$ is given by

$$(3.28) \quad n_{\varphi,i} = 2n_\theta \sin(\theta_i) + 1.$$

Based on this sampling, let $\mathbf{v}_b$ be the vector pointing to a sampling point and corresponding to the histogram bin b . Then the histogram is filled with the weights w_b , which are calculated for the normal vector $\mathbf{n}_i$ of each point $\mathbf{p}_i$ within an $\mathcal{E}$ -neighborhood $\mathcal{N}_{\mathbf{p},\mathcal{E}}$ of a point $\mathbf{p}$ with

$$(3.29) \quad w_b = \begin{cases} 0 & , \text{ if } \mathbf{n}_i \cdot \mathbf{v}_b < \cos \alpha, \\ \frac{\mathbf{n}_i \cdot \mathbf{v}_b - \cos \alpha}{1 - \cos \alpha} & , \text{ otherwise.} \end{cases}$$

The angle α denotes the maximum angle to be used to compute the histogram. Finally, the histogram for point $\mathbf{p}$ is normalized and the entropy is calculated according to Equation 3.27.

To select keypoints from those entropy values, Fiolka et al. first define a threshold to skip all points with an entropy below. This is done to handle noisy data which could cause local maxima of the surface entropy. In the next step they test for a considerable variance of surface entropy in all directions to find corners and edges, beginning with the computation of the local center of entropy mass within a neighborhood $\mathcal{N}_{\mathbf{p},\mathcal{E}}$ for a point $\mathbf{p}$

$$(3.30) \quad \mu_H(\mathcal{N}_{\mathbf{p},\mathcal{E}}) = \frac{1}{\sum_{\mathbf{p}_i \in \mathcal{N}_{\mathbf{p},\mathcal{E}}} h(X_{\mathcal{N}_{\mathbf{p}_i,\mathcal{E}}})} \sum_{\mathbf{p}_i \in \mathcal{N}_{\mathbf{p},\mathcal{E}}} h(X_{\mathcal{N}_{\mathbf{p}_i,\mathcal{E}}}) \mathbf{p}_i.$$

This enables the computation of the entropy covariance matrix:

$$(3.31) \quad \mathbf{C}_{H(\mathcal{N}_{\mathbf{p},\mathcal{E}})} = \frac{1}{\sum_{\mathbf{p}_i \in \mathcal{N}_{\mathbf{p},\mathcal{E}}} h(X_{\mathcal{N}_{\mathbf{p}_i,\mathcal{E}}})} \sum_{\mathbf{p}_i \in \mathcal{N}_{\mathbf{p},\mathcal{E}}} h(X_{\mathcal{N}_{\mathbf{p}_i,\mathcal{E}}}) \cdot \left((\mathbf{p}_i - \mu_H(\mathcal{N}_{\mathbf{p},\mathcal{E}})) (\mathbf{p}_i - \mu_H(\mathcal{N}_{\mathbf{p},\mathcal{E}}))^T \right).$$

By an eigenvalue decomposition of $\mathbf{C}_{H(\mathcal{N}_{\mathbf{p},\mathcal{E}})}$ the final decision in favor of or against $\mathbf{p}$ as keypoint is

$$(3.32) \quad \frac{\lambda_1}{\lambda_3} \geq p_{min}.$$

Fiolka et al. recommend a $p_{min} = 0.15$.

3.2 Performance Evaluation

For a selection of suitable keypoint algorithms among those presented so far, already existing evaluations of all these methods would be helpful. However, at this moment only a handful of reviews exists.

Bronstein et al. [8] presented a review of algorithms used for 3-D recognition and classification tasks based on local features in 2010. In their work, however, they present only four keypoint algorithms, whose application domain is primarily limited to meshes.

Sali et al. [72] presented a performance evaluation just for keypoint detection algorithms in 2011. Of the eight proposed methods, at least five are suitable for depth images and point cloud. These works are local surface patches (LSP) from Chen and Bhanu (see Section 3.1.4), the work of Unnikrishnan and Hebert with a scale space based on the Laplace-Beltrami operator (LBSS) (see Section 3.1.6), the intrinsic shape signature (ISS) by Zhong (see Section 3.1.8), and two variants (KPQ & KPQ-SI) of the keypoint detection algorithms from Mian et al., where KPQ-SI is the proposed scale invariant version (see Section 3.1.9).

Salti et al. use two synthetic and two captured datasets in their experiments. They define the quality of a keypoint detector by the ability of the detector to find the same set of keypoints for different views of the same part of an object, what they name *repeatability*. In addition, they vary the scenes regarding occlusion. To simulate sensor noise, they add 3 levels of Gaussian noise on the two synthetic datasets.

The results of these experiments can be summarized as follows:

- The ISS keypoint method by Zhong is by far the fastest algorithm proposed. However, this method is not scale invariant, so that its application may be problematic on data from different sources and with different scales.
- The adaptive KPQ-SI keypoint method by Mian et al. has relatively high repeatability rates and is relatively reliable on noisy data. For this, however, computation times, which are one order of magnitude greater than those of ISS must be accepted.

Dutagaci et al. [18] chose another approach to evaluate the quality of keypoint detectors. They created a website and asked visitors of the website to mark points of interest on the displayed 3-D objects. The results were transferred into a ground truth, first by combining all markers within a radius r to a single group. Then all groups selected by less than n users were discarded. Finally they selected a representative point for each group with the minimum sum of geodesic distances to all other points within the group. This ground truth was compared to the results of 6 keypoint detection algorithms. However, all the methods used in the article are intended for meshes.

In 2013 Filipe and Alexandre [19] compared four keypoint algorithms with a focus on implementations available in the Point Cloud Library (PCL) [68]. The algorithms are 3-D extensions of the Harris corner detector [28] and SUSAN [77], where the image gradients in the covariance matrix get replaced by surface normals, as well as Thrift (see Section 3.1.5) and ISS (see Section 3.1.8). Like Sali et al., Filipe and Alexandre prefer the repeatability as a quality measure. They conclude that Thrift and ISS yielded the best scores in terms of repeatability. They also note that ISS is the fastest of the tested algorithms.

Gou et al. [27] published 2014 a comprehensive survey paper with focus on 3-D object recognition based on local features. Among other aspects of this application area they present a total of 29 keypoint detection methods for meshes, depth images and point clouds. The mostly short reviews of the proposed algorithms, however, do not come from their own evaluations.

4

3-D Object Description

One of the main components of the basic 3-D classification pipeline is the computation of a distinctive description of the point cloud. It must be considered that the point cloud may be partially occluded. Accordingly, in most cases, local feature descriptors are used, which are summarized to a global description in a subsequent step. This chapter begins with numerous different algorithms for local feature descriptions. Afterwards it is described how, in the context of this work, a so-called bag of features approach can be used to combine the local feature descriptions to get an aggregated representation that can be used as a global description of the point cloud.

4.1 Local Feature Descriptors

The goal of local descriptors is the description of particularly “interesting” local areas of a 3-D object. In contrast to global 3-D descriptors not just one but rather several signatures or histograms will be computed. The advantages of local representations are, that they are robust with respect to noise, variability in object shape and partial occlusions [27]. On the other hand they remain less discriminating due to the limited scope of the local neighborhood. For this reason, the subsequent comparison of feature descriptions is another challenging part of a recognition pipeline which will be discussed later in this chapter.

The algorithms discussed below are distinguished with respect to the following characteristics:

- Is the method designed to work with point clouds and/or depth images?
- Does the method compute a signature or a histogram?

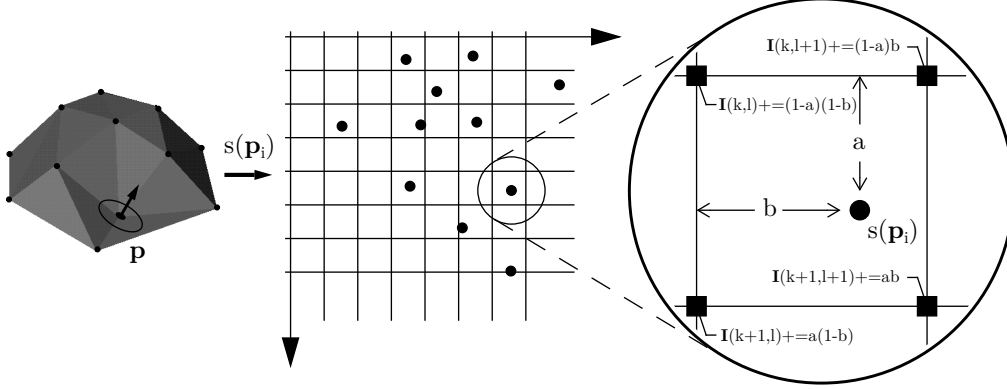


Figure 4.1: The figure from [38] depicts the calculation of spin images. The values a and b are: $a = (\dot{\mathbf{p}}_y - \delta_y \cdot k) / \delta_y$ and $b = (\dot{\mathbf{p}}_x - \delta_x \cdot l) / \delta_x$

- Does the method require or compute a local reference frame, i. e., a normal vector or the principal axis of the local part of the point cloud?

4.1.1 Spin Images

The spin image descriptor introduced 1999 by Johnson and Hebert [38, 37] is arguably the most cited and popular local 3-D descriptor. It is a histogram based method that requires a normal vector as a rotation axis. The algorithm was introduced for 3-D surfaces, but it also performs well on point clouds if the computation of a normal vector is possible.

For the determination of the spin images Johnson and Hebert first introduce the notion of a spin map. The spin map s maps the 3-D points $\mathbf{p} \in \mathbb{R}^3$ to 2-D points $\dot{\mathbf{p}} \in \mathbb{R}^2$ in the following manner:

$$(4.1) \quad s(\mathbf{p}_i) = \dot{\mathbf{p}}_i = \left(\sqrt{\|\mathbf{p}_i - \mathbf{p}\|^2 - (\mathbf{n} \cdot (\mathbf{p}_i - \mathbf{p}))^2}, \mathbf{n} \cdot (\mathbf{p}_i - \mathbf{p}) \right)^\top,$$

where $\mathbf{p}_i \in \mathcal{P}$ is an arbitrary point of the point cloud, $\mathbf{p} \in \mathcal{P}$ is the 3-D point for which the spin image is to be calculated, and $\mathbf{n}$ is the normal vector of $\mathbf{p}$. This leads to 2-D points, where $\dot{\mathbf{p}}_x$ cannot be negative.

The spin image $\mathbf{I}$ is a matrix of size $n \times m$. Each matrix element has a size of δ_x and δ_y . The values of this matrix are determined based on the spin map, allocating a value of 1 for each mapped 2-D point $\dot{\mathbf{p}}$ to the 4 corresponding matrix elements (k, l) , $(k+1, l)$, $(k+1, l+1)$, and $(k, l+1)$ of $\mathbf{I}$ by bilinear interpolation,

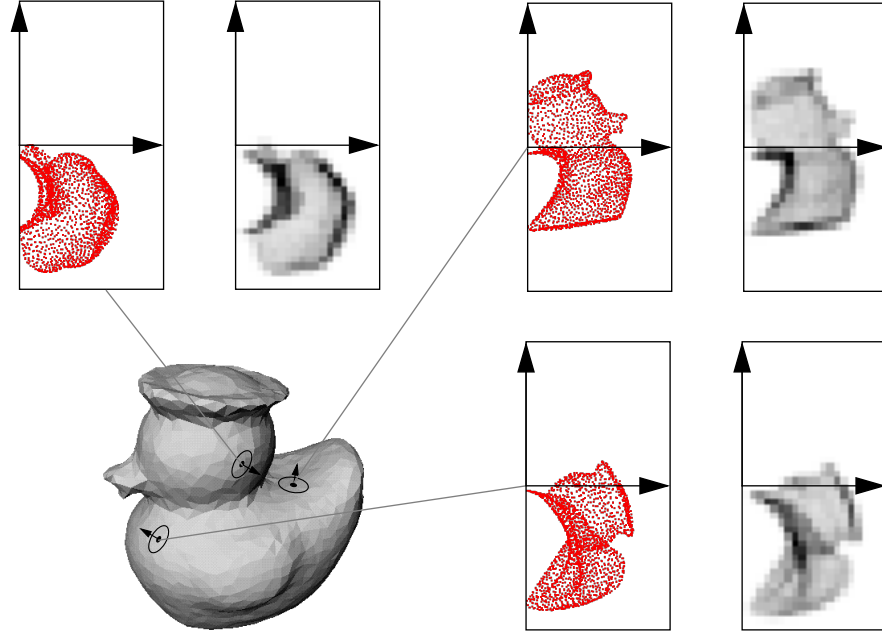


Figure 4.2: This figure from [38] represents 2-D point clouds (red) and the corresponding spin image (gray) for three exemplary selected 3-D points.

where $l \cdot \delta_x \leq \dot{\mathbf{p}}_x$, $(l + 1) \cdot \delta_x \geq \dot{\mathbf{p}}_x$, $k \cdot \delta_y \leq \dot{\mathbf{p}}_y$, and $(k + 1) \cdot \delta_y \geq \dot{\mathbf{p}}_y$. This is shown in Figure 4.1.

A spin images can also be represented as a normalized gray-scale image. This allows a simple visual check of the local descriptions (see Figure 4.2).

4.1.2 3-D Shape Context

The descriptor called 3-D shape context proposed by Frome et al. [23] in 2004 is a histogram based descriptor for 3-D point clouds. The spherical histogram requires a normal vector for the alignment of its north pole, but it does not have a unique reference frame.

Let $\mathbf{p} \in \mathcal{P}$ be the point for which a local description is to be determined. Let $\mathbf{n}$ be the normal of $\mathbf{p}$. Let S be a sphere with its north pole aligned to $\mathbf{n}$. The sphere is divided into $J + 1$ radial divisions, $K + 1$ elevation divisions, and $L + 1$ azimuth divisions. Each bin accumulates a weighted count of all points that fall within its coordinate intervals. The weight of each point is given by

$$(4.2) \quad \omega(\mathbf{p}_i) = \frac{1}{\rho_i \sqrt[3]{V(j, k, l)}},$$

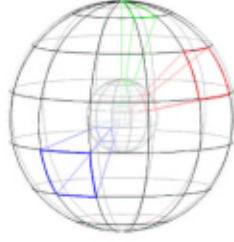


Figure 4.3: The figure from [23] shows a schematic view of the 3-D shape context histogram.

where $V(j, k, l)$ is the volume of the bin and ρ_i is the local point density, which is estimated by the number of points that fall into a sphere with radius r around point $\mathbf{p}_i$. The values of this histogram result in a $J \times K \times L$ -dimensional feature vector. A schematic visualization of this histogram is shown in Figure 4.3.

As already mentioned, the 3-D shape context does not have a unique reference frame. The problem is that the azimuth angle is not bound. Frome et al. overcome this problem by calculating additional feature vectors for the searched object, using each azimuth segment once as a reference for the azimuth angle.

4.1.3 Local Surface Patches

Chen and Bhanu [10] introduced an integrated local surface descriptor in 2007. Their approach is intended to be used on depth images and determines a histogram. Since the histogram is build upon the angles between normal vectors it does not require a (uniform) local reference frame.

Chen and Bhanu name the local region around a keypoint $\mathbf{p} \in \mathcal{P}$ a 'local surface patch'. Let $\mathcal{N}_{\mathbf{p}} \subseteq \mathcal{P}$ be the neighborhood of $\mathbf{p}$ with

$$(4.3) \quad \mathcal{N}_{\mathbf{p}} = \left\{ \mathbf{p}_i | \mathbf{p}_i \in \mathcal{P}, \|\mathbf{p}_i - \mathbf{p}\| < \epsilon, \cos^{-1}(\mathbf{n}_{\mathbf{p}_i} \cdot \mathbf{n}_{\mathbf{p}}) \in [-1, 1] \right\},$$

where $\mathbf{n}_{\mathbf{p}_i}$ and $\mathbf{n}_{\mathbf{p}}$ are normal vectors of $\mathbf{p}_i$ and $\mathbf{p}$. The elements of $\mathcal{N}_{\mathbf{p}}$ are limited to an Euclidean distance ϵ and the angle between the normal vectors $\mathbf{n}_{\mathbf{p}_i}$ and $\mathbf{n}_{\mathbf{p}}$. In addition to the angle, Chen and Bhanu compute a quantitative measure of the surface named 'shape index' [16] for each $\mathbf{p}_i \in \mathcal{N}_{\mathbf{p}}$:

$$(4.4) \quad s(\mathbf{p}_i) = \frac{1}{2} - \frac{1}{\pi} \tan^{-1} \frac{\kappa_1(\mathbf{p}_i) + \kappa_2(\mathbf{p}_i)}{\kappa_1(\mathbf{p}_i) - \kappa_2(\mathbf{p}_i)}.$$

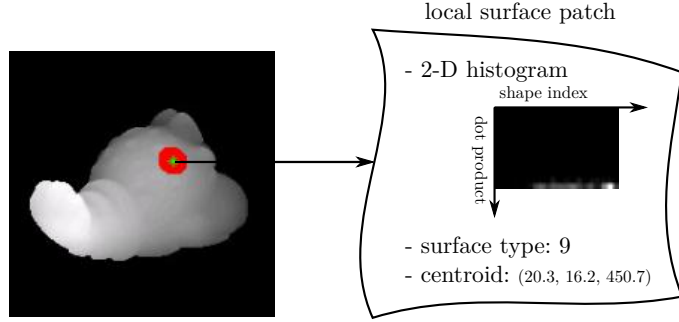


Figure 4.4: The figure is adopted from [10] and shows a depth image with its keypoint (green) and the local surface patch (red) on the left side. On the right side a schematic view of the descriptor results, i. e., the histogram, the surface type, and the centroid are shown.

The angle and the shape index form the two axes of a histogram. The histogram is filled in the same way as Johnson and Hebert do for spin images (see Section 4.1.1).

In addition to the histogram, they determine a value, which they name 'surface type':

$$(4.5) \quad t = 1 + 3(1 + \text{sgn}(H)) + (1 - \text{sgn}(K)),$$

where H is a mean curvature and K is a Gaussian curvature.

Finally they use a combination of the means and standard deviations as keys for a hash table, while the values consist of the model id, the surface type, the centroid of the patch and the histogram. This is illustrated in Figure 4.4.

4.1.4 Thrift

Flint et al. [22] proposed a descriptor for point clouds in 2007 they named 'Thrift'. It is a histogram based descriptor which does not require a local reference frame, since each histogram represents the number of pairs of normal vectors with similar angles to each other.

Flint et al. define the local neighborhood $\mathcal{N}_{\mathbf{p}} \subseteq \mathcal{P}$ for a keypoint $\mathbf{p} \in \mathcal{P}$ as

$$(4.6) \quad \mathcal{N}_{\mathbf{p}} = \{\mathbf{p}_i | \mathbf{p}_i \in \mathcal{P}, \|\mathbf{p}_i - \mathbf{p}\| \leq \sigma\},$$

what they name 'support'. For each element of $\mathbf{p}_i \in \mathcal{N}_{\mathbf{p}}$ they define two windows:

$$(4.7) \quad \begin{aligned} \mathcal{W}_{\text{small},i} &= \{\mathbf{p}_j | \mathbf{p}_j \in \mathcal{P}, \|\mathbf{p}_j - \mathbf{p}_i\| \leq \omega_{\text{small}}\}, \\ \mathcal{W}_{\text{large},i} &= \{\mathbf{p}_j | \mathbf{p}_j \in \mathcal{P}, \|\mathbf{p}_j - \mathbf{p}_i\| \leq \omega_{\text{large}}\}. \end{aligned}$$

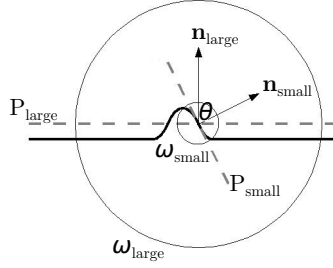


Figure 4.5: The figure is adopted from [22] and shows the planes P_{small} and P_{large} with the corresponding normals $\mathbf{n}_{\text{small},i}$ and $\mathbf{n}_{\text{large},i}$ for an exemplary point on a sample surface. The bin of the descriptor histogram is selected by the angle θ .

These windows are used to approximate two planes P_{small} and P_{large} with the method of least squares. The normal vectors of these planes are $\mathbf{n}_{\text{small},i}$ and $\mathbf{n}_{\text{large},i}$ as shown in Figure 4.5

A bin of a histogram is selected depending on the angle θ between the normal vector pair $\mathbf{n}_{\text{small},i}$ and $\mathbf{n}_{\text{large},i}$ and increased by one for all points in the neighborhood. Finally, the histogram is normalized to a sum of 1.

The number of bins and the parameters ω_{small} and ω_{large} are user defined parameters. Flint et al. use 10 bins, $\omega_{\text{small}} = 0.3\sigma$, and $\omega_{\text{large}} = 0.8\sigma$.

4.1.5 Point Feature Histogram

Rusu et al. [66] proposed a 'point feature histogram' (PFH) in 2008. As its name already suggests, it is a histogram based approach. PFH does not require a local reference frame. The algorithm can be used on depth images, as well as on point clouds. However, it is assumed, that a viewpoint is known.

Rusu et al. compute a Darboux frame to define a descriptor. Let $\mathcal{N}_{\mathbf{p}} \subseteq \mathcal{P}$ be the local neighborhood of a keypoint $\mathbf{p} \in \mathcal{P}$ with

$$(4.8) \quad \mathcal{N}_{\mathbf{p}} = \{\mathbf{p}_i | \mathbf{p}_i \in \mathcal{P}, \|\mathbf{p}_i - \mathbf{p}\| \leq r\},$$

where r is the radius of the spherical neighborhood. Each point $\mathbf{p}_i \in \mathcal{P}$ requires a corresponding normal vector $\mathbf{n}_i$, which may have to be approximated using a principal component analysis for $\mathcal{N}_{\mathbf{p}_i}$. To get consistent normal vectors they re-orient them with respect to the viewpoint $\mathbf{c}$:

$$(4.9) \quad \langle \mathbf{c} - \mathbf{p}_i, \mathbf{n}_i \rangle < 0 : \mathbf{n}_i = -\mathbf{n}_i.$$

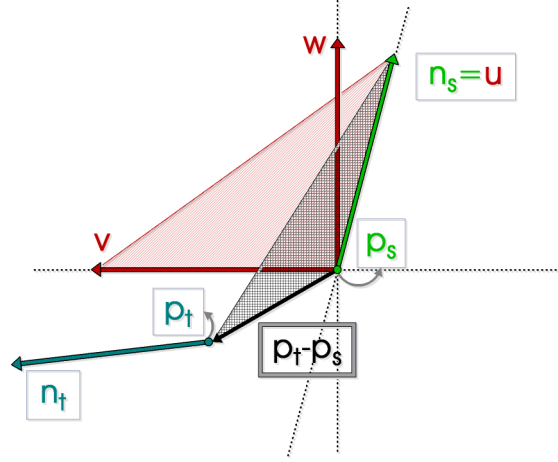


Figure 4.6: This figure from [66] shows a Darboux frame consisting of the vectors $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$.

Furthermore, let $\mathbf{p}_i, \mathbf{p}_j \in \mathcal{N}_{\mathbf{p}}, i \neq j, j < i$ a pair of points within the neighborhood of $\mathbf{p}$. One of the two points are the source point $\mathbf{p}_s$, the other the target point $\mathbf{p}_t$:

$$(4.10) \quad \begin{array}{ll} \text{if} & \langle \mathbf{n}_i, \mathbf{p}_j - \mathbf{p}_i \rangle \leq \langle \mathbf{n}_j, \mathbf{p}_i - \mathbf{p}_j \rangle, \\ \text{then} & \mathbf{p}_s = \mathbf{p}_i, \mathbf{p}_t = \mathbf{p}_j, \\ \text{otherwise} & \mathbf{p}_s = \mathbf{p}_j, \mathbf{p}_t = \mathbf{p}_i. \end{array}$$

Then the Darboux frame (see Figure 4.6) is defined by

$$(4.11) \quad \mathbf{u} = \mathbf{n}_s, \mathbf{v} = (\mathbf{p}_t - \mathbf{p}_s) \times \mathbf{u}, \mathbf{w} = \mathbf{u} \times \mathbf{v}.$$

Finally Rusu et al. define four values build upon the Darboux frame and the point pairs:

$$(4.12) \quad \begin{aligned} f_1 &= \langle \mathbf{v}, \mathbf{n}_t \rangle, \\ f_2 &= \|\mathbf{p}_t - \mathbf{p}_s\|, \\ f_3 &= \langle \mathbf{u}, \mathbf{p}_t - \mathbf{p}_s \rangle / f_2, \\ f_4 &= \tan^{-1} \langle \mathbf{w}, \mathbf{n}_t \rangle. \end{aligned}$$

The values of f_1 and f_3 range from -1 to 1 , the values of f_2 are between 0 and $2 \cdot r$, and the values of f_4 are between $\pm\pi/2$. Let $b(s, f)$ be a binary function that is 0 if $f < s$ and 1 otherwise. Then

$$(4.13) \quad i = b(0, f_4) \cdot 2^3 + b(0, f_3) \cdot 2^2 + b(r, f_2) \cdot 2^1 + b(0, f_1)$$

is an index between 0 and 15, that are used to increase one of 16 bins of a histogram for each point pair of $\mathcal{N}_{\mathbf{p}}$. Finally, the bins of the histogram are normalized so that the sum of all bins is 1.

In later publications Rusu et al. note, that two significant changes lead to better results [71, 65]:

- The length feature (f_2 in Equation 4.12) is omitted, since it has been found that this feature does not only not increase robustness, but causes problems in some situations in depth images.
- The subdivision intervals in the feature's value range are increased from 2 to 5. This results in a 125-D feature vector in place of the 16-D vector.

4.1.6 Fast Point Feature Histogram

Since the computational complexity for the determination of a Darboux frame at each point with a k -neighborhood is $O(k^2)$, the computation of point feature histograms is relatively slow. For that reason, Rusu et al. [71] proposed a simplified version of PFH in 2009 they named 'fast point feature histogram' (FPFH). They preserved the basic characteristics of the descriptor, but changed the computation of the Darboux frame with an approximation of it. The changes from PFH to FPFH are as follows:

- The Darboux frame and the three remaining features (Equation 4.12 without the distance feature f_2) will no longer be estimated for point pairs in the neighborhood $\mathcal{N}_{\mathbf{p}}$ of a keypoint $\mathbf{p}$, but for each point $\mathbf{p}' \in \mathcal{P}$ and k points $\mathbf{p}'_i$ in a k -neighborhood of $\mathbf{p}'$. Thus, for each point of the point cloud a reduced histogram is filled, which Rusu et al. call a 'simplified point feature histogram' (SPFH).
- In a second iteration they compute the FPFH (see Figure 4.7) for a keypoint $\mathbf{p}$ with the SPFHs for each point $\mathbf{p}_i$ in the neighborhood $\mathcal{N}_{\mathbf{p}}$ weighted by the distance $\omega_i = \|\mathbf{p} - \mathbf{p}_i\|$:

$$(4.14) \quad FPFH(\mathbf{p}) = SPFH(\mathbf{p}) + \frac{1}{|\mathcal{N}_{\mathbf{p}}|} \sum_{\mathbf{p}_i \in \mathcal{N}_{\mathbf{p}}} \frac{1}{\omega_i} SPFH(\mathbf{p}_i).$$

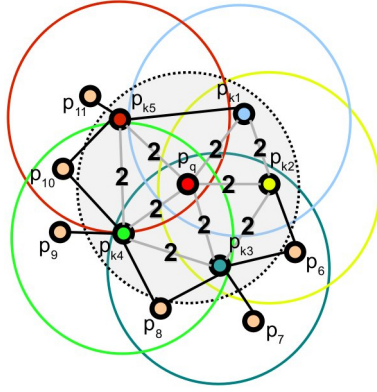


Figure 4.7: This figure from [71] shows the relationship between the simplified point feature histograms and the fast point feature histogram. The red point $\mathbf{p}_q$ in the center of this figure is the keypoint for which the FPFH is determined. The FPFH is combined of SPFH (corresponding to the colored circles) of all points $\mathbf{p}_{k1}, \dots, \mathbf{p}_{k5}$ in the neighborhood $\mathcal{N}_{\mathbf{p}_q}$ represented by the dotted circle.

- To reduce the dimensionality (125 dimensions for a PFH) and the often empty bins, they propose the decorrelation of the values. The values of the three features are filled into separate feature histograms with 11 bins and then concatenated to a 33-D feature description.

4.1.7 2.5-D SIFT

The algorithm proposed by Lo and Siebert [48] in 2009 is a feature descriptor for depth images. The algorithm is inspired by the 2-D version of SIFT and computes a histogram based on oriented gradients. A uniform reference frame is required and computed.

The reference frame is determined by the tuple $(x, y, \sigma, \theta, \varphi, \tau)$ for each keypoint. The values of x, y and σ are the in-plane coordinates and the scale of the keypoint $\mathbf{p}$. The values of x and y are taken from the keypoint used. The angle θ is a gradient orientation. Its calculation was presented within Section 3.1.7 of the 2.5-D SIFT keypoint algorithm. The computation of the other values requires a surface normal $\mathbf{n}$ for which they use an approach proposed by Sze et al. [86]:

$$(4.15) \quad \mathbf{n} = \frac{(-f(\mathbf{p})_x, -f(\mathbf{p})_y, 1)^\top}{\sqrt{f(\mathbf{p})_x^2 + f(\mathbf{p})_y^2 + 1}},$$

where $f(\mathbf{p})_x$ and $f(\mathbf{p})_y$ are the first Gaussian derivatives of $\mathbf{p}$ in x and y direction, using the known scale σ . With that normal vector Lo and Siebert compute the two missing angles of the slant φ and the tilt τ :

$$\varphi = \tan^{-1} \left(\frac{\sqrt{\mathbf{n}_x^2 + \mathbf{n}_y^2}}{\mathbf{n}_z} \right), \quad (4.16)$$

$$\tau = \tan^{-1} \left(\frac{\mathbf{n}_x}{\mathbf{n}_y} \right).$$

The computation of the descriptor values is done in four steps:

1. The area of the local surface patch around the keypoint is rotated with respect to the canonical in-plane orientation θ . Then the degree of curvature is calculated based on the Gaussian curvature K and the mean curvature H :

$$c = \sqrt{2H^2 - K}.$$
2. The local surface patch is divided into nine elliptical sub-regions, overlapping by one standard deviation, as shown in Figure 4.8. The ellipses have an aspect ratio of φ and an orientation given by τ .
3. Two different histograms are filled for each of the elliptical sub-regions. On the one hand they create a histogram $H_{i_{\text{surface}}}$ of nine surface types (from cup to cap) based on the local curvature values and weighted by c . On the other hand they formulate an eight bin histogram $H_{i_{\text{orientation}}}$ covering the range of all possible orientations. Both histograms are normalized to a magnitude of 1 and concatenated to one of nine local histograms:

$$H_i = [H_{i_{\text{surface}}}, H_{i_{\text{orientation}}}]$$

4. All nine histograms build the final descriptor:

$$D = [H_1, \dots, H_9].$$

4.1.8 Intrinsic Shape Signatures

The 'intrinsic shape signatures' introduced by Zhong [105] in 2009 is a histogram based feature descriptor for point clouds. The method includes the determination of a reference frame, but its orientation is ambiguous.

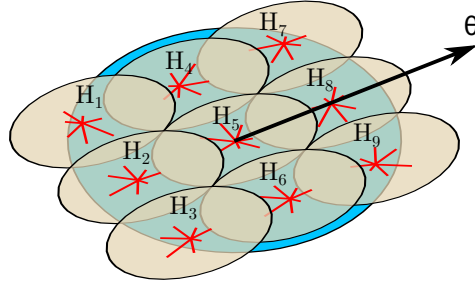


Figure 4.8: This is an illustration of the local surface patch (blue), which is oriented by θ and divided into nine elliptical sub-regions (beige). For each sub-region weighted a histogram H_i is filled with surface types and orientations.

As already mentioned in the summary of the keypoint extraction algorithm in Section 3.1.8, Zhong computes the eigenvalues and eigenvectors of a spherical neighborhood. The eigenvectors $\mathbf{e}_1$, $\mathbf{e}_2$ and the cross product $\mathbf{e}_1 \times \mathbf{e}_2$ is used as x -, y - and z -axis for four possible local reference frames.

Similar to 3-D shape context from Frome et al. (see Section 4.1.2) the intrinsic shape signature (ISS) is a spherical histogram with radius r . But instead of a segmentation based on the spherical radius, the azimuth, and the elevation, Zhong computes a spherical grid starting with an octahedron. First, she divides each triangle into 4 smaller triangles by adding new vertexes at the midpoint of each edge. Each new vertex is moved along an axis from the center to the vertex, until it intersects the surface of a surrounding sphere. In a third step each point of the sphere is assigned to its nearest neighbor, which leads to a Voronoi diagram on the surface (see Figure 4.9). These three steps can be repeated multiple times to refine the cells on the sphere. After three iteration the spherical tessellation consists of $n = 66$ Voronoi cells. This process leads to uniformly and homogeneously distributed cells.

To ensure that the algorithm is able to select the corresponding cell as quickly as possible, Zhong stores a lookup table to map each angle pair (θ, φ) to the corresponding bin. In combination with l radial distances $\rho_0, \rho_1, \dots, \rho_{l-1}$ she partitions the spherical volume into $k = 1 + (l - 1) \cdot n$ bins with no angular discrimination for radii $\rho < \rho_0$. Each point $\mathbf{p}_i$ that falls into a bin is weighted by $w_i = 1/|\{\mathbf{p}_j : \|\mathbf{p}_j - \mathbf{p}_i\| < r\}|$, the inverse of the amount of points in a spherical neighborhood of radius r . The sums within the bins form the feature vector.

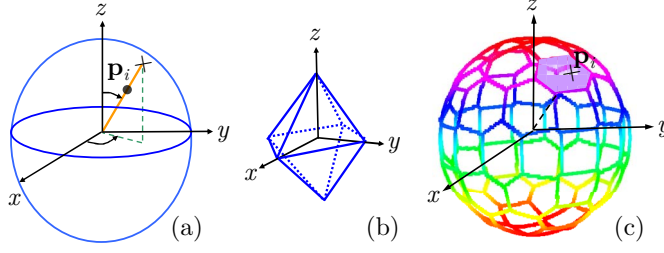


Figure 4.9: These figures of spherical tessellation as a method for binning are adopted from [105]: (a) a point p_i and its axis with the associated angle pair (θ, φ) , (b) the base structure, an octahedron, is used for spherical tessellation, which results in (c) a spherical Voronoi grid. The cell corresponding to (θ, φ) is filled in purple.

4.1.9 Shape Index SIFT

Bayramoglu and Alatan [6] proposed a feature descriptor for depth images in 2010. The approach is straightforward. They compute the shape indexes as shown in Equation 4.4 for all points of the depth image and map the values to a gray scale image in the range of 0-255. Then they apply the 2-D SIFT from Lowe [49] on the gray scale image.

4.1.10 Surface Fitting with a Uniform Lattice

The method proposed by Mian et al. [56] in 2010 determines a signature for meshes, point clouds and depth images in the same way. It does not have a unique reference frame, but requires an already aligned partition of the surface $\hat{\mathcal{M}}_i$ as determined by the keypoint detection part of their work mentioned in Section 3.1.9.

To determine the feature description a surface fitting of a surface $\mathcal{S}$ onto the point cloud is performed using a MATLAB extension named 'gridfit' [15]. The extension is applied with a smoothing factor to get a dense smoothed surface without sharp edges and with damped noise. Furthermore, they define a uniform $n \times n$ grid with $n = 20$, which they align to the x - and y -axis of the local reference frame of $\hat{\mathcal{M}}_i$ and $\mathcal{S}$, respectively. The grid is used to sample the relative depth values of the surface $\mathcal{S}$.

Mian et al. fill a 400 dimensional vector with these grid values. However, as a dimensionality of 400 is quite high, they reduce the vector size using a singular value decomposition of the covariance matrix made from all vectors of a set of

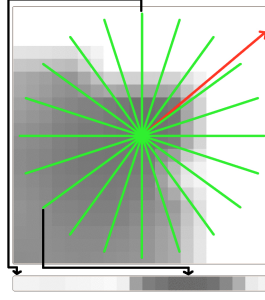


Figure 4.10: This figure from [79] shows the star pattern placed on top of the smoothed grid of depth values used to compute the NARF description.

scanned 3-D objects. They propose that only the highest $k = 10$ eigenvalues should to be used to map the 400 dimensional vector to a 10 dimensional description. Due to their experiments they propose a fidelity of 95% for these low dimensional feature descriptions.

4.1.11 NARF

Steder et al. [79, 80] proposed a signature based feature descriptor for depth images in 2010. The algorithm creates a local reference frame for each feature point and its local environment.

The keypoint defines the origin of the local reference frame. The z -axis is parallel to the normal vector and the y axis is oriented along the upright vector by definition. The area with a radius of $\sigma/2$ is translated to the local reference frame and rotated in such a way, that the dominant direction α_i , calculated with the corresponding keypoint detection algorithm described in Section 3.1.11, is aligned to the y -axis.

Based on the aligned patch with radius $\sigma/2$, Steder et al. use a 10×10 grid oriented to the x - and y -axes of the local reference frame. The grid is filled with the smallest z -values of all points falling into each cell. Cells without a corresponding 3-D point get filled with the maximum value of the local area. Finally, the patch is smoothed with a gaussian blur.

Based on the smoothed patch, Steder et al. extract the descriptor values d_i placing a star pattern with n beams on top of this grid (see Figure 4.10). Let

$c_{i,0}, \dots, c_{i,m}$ be the cells of the patch lying under beam b_i starting with $c_{i,0}$ at the center of the star pattern. Then d_i is calculated by

$$\begin{aligned}
 w(c_{i,j}) &= 2 - \frac{2 \cdot \|c_{i,j} - c_{i,0}\|}{\sigma}, \\
 d'_i &= \frac{\sum_{j=0}^{m-1} (w(c_{i,j}) \cdot (c_{i,j+1} - c_{i,j}))}{\sum_{j=0}^{m-1} w(c_{i,j})}, \\
 d_i &= \frac{\text{atan2}(d'_i, \frac{\sigma}{2})}{180^\circ}.
 \end{aligned}
 \tag{4.17}$$

Here, $w(c_{i,j})$ is a distance based weighting factor, whereby changes of the surface nearby the center affect the resulting values more than surface changes at a larger distance. By using atan2 the values are normalized to a range of $[-0.5, 0.5]$. So far, this descriptor is not invariant to the rotation. Thus, Steder et al. use a histogram with bins for each possible orientation in steps of one degree. The value of each bin for an orientation β is

$$h(\beta) = \frac{1}{2} + \frac{1}{n} \sum_{i=1}^n d_i \cdot \left(1 - \frac{|\beta - \gamma_i|}{180^\circ}\right)^2,
 \tag{4.18}$$

where γ_i is the angle of the i -th beam. Finally, the dominant orientation and all orientations with a value exceeding 80% of the maximum value are used to store the corresponding description.

4.1.12 Signatures of Histograms of Orientations

Tombari, Salti and Di Stefano [93, 73] introduced a 3-D point cloud descriptor algorithm in 2010 they dub 'signatures of histograms of orientations' (SHOT). The algorithm is designed to handle depth images, point clouds and meshes. Furthermore, they proposed an extension in 2011 to use RGB data if it is available [94]. A major part of their work devotes attention to the estimation of a stable local reference frame and they state that the local reference frame is more stable in comparison to other recent approaches, particularly to the two local feature description algorithm for meshes called MeshHOG [104] and 3-D SURF [41].

Similar to other already mentioned approaches they compute a covariance matrix $\mathbf{C}$ of the local neighborhood with a radius r of a point $\mathbf{p}$ to estimate a

normal vector by eigenvalue decomposition. To improve the robustness against clutter, they weight the points by distance

$$(4.19) \quad \mathbf{C} = \frac{1}{\sum_{i, d_i \leq r} (r - d_i)} \sum_{i=1}^k (r - d_i) (\mathbf{p}_i - \mathbf{p})(\mathbf{p}_i - \mathbf{p})^\top, \quad d_i = \|\mathbf{p}_i - \mathbf{p}\|_2.$$

Let $\mathbf{x}^+$, $\mathbf{y}^+$, and $\mathbf{z}^+$ be the unit vectors with their directions pointing to the eigenvectors corresponding to the eigenvalues in descending order. Let $\mathbf{x}^-$, $\mathbf{y}^-$, and $\mathbf{z}^-$ be the unit vectors in the opposite direction. The necessary steps to decide which of the two conceivable unit vectors should be used for each direction is shown exemplary for the x -axis:

Initially Tombari, Salti and Di Stefano compute two sets of points that contain all points $\mathbf{p}_i$ which lie in front of $\mathbf{p}$ with respect to $\mathbf{x}^+$ and $\mathbf{x}^-$

$$(4.20) \quad \begin{aligned} \mathcal{S}_x^+ &= \{\mathbf{p}_i : d_i \leq r \wedge (\mathbf{p}_i - \mathbf{p})\mathbf{x}^+ \geq 0\}, \\ \mathcal{S}_x^- &= \{\mathbf{p}_i : d_i \leq r \wedge (\mathbf{p}_i - \mathbf{p})\mathbf{x}^- > 0\}. \end{aligned}$$

If one of the two sets contains more points than the other, the direction of the x -axis points to the side of the most points, i.e., $\mathbf{x}^+$ if $|\mathcal{S}_x^+| > |\mathcal{S}_x^-|$ or $\mathbf{x}^-$ if $|\mathcal{S}_x^+| < |\mathcal{S}_x^-|$. In the (rare) case that $|\mathcal{S}_x^+| = |\mathcal{S}_x^-|$, they estimate the median distance d_m and select those k points $\mathbf{p}_i$ within the neighborhood of $\mathbf{p}$ whose distance $d_i = \|\mathbf{p}_i - \mathbf{p}\|_2$ is the closest to d_m :

$$(4.21) \quad \mathcal{M}(k) = \left\{ \mathbf{p}_i : \min_k (|d_m - d_i|) \right\}.$$

Based on this set they compute two further sets

$$(4.22) \quad \begin{aligned} \tilde{\mathcal{S}}_x^+ &= \{\mathbf{p}_i \in \mathcal{M}(k) \wedge (\mathbf{p}_i - \mathbf{p})\mathbf{x}^+ \geq 0\}, \\ \tilde{\mathcal{S}}_x^- &= \{\mathbf{p}_i \in \mathcal{M}(k) \wedge (\mathbf{p}_i - \mathbf{p})\mathbf{x}^- > 0\} \end{aligned}$$

and chose the direction of the x -axis in the same manner as above: $\mathbf{x}^+$ if $|\tilde{\mathcal{S}}_x^+| > |\tilde{\mathcal{S}}_x^-|$ or $\mathbf{x}^-$ otherwise. This procedure is done in the same way for the z -axis. Finally, $\mathbf{x} \times \mathbf{z}$ results in $\mathbf{y}$.

To compute the SHOT description Tombari, Salti and Di Stefano use a spherical segmentation as shown in Figure 4.11. For each for the segments they compute a histogram filled with the cosine values of the angles between the z -axis of the local reference frame and the normal vectors $\mathbf{n}_j$ of points $\mathbf{p}_j$ which are part of the currently considered segment, i.e., $\mathbf{z}_\mathbf{p} \cdot \mathbf{n}_j$. To avoid boundary effects, they perform a quadrilinear interpolation with the neighboring bins and with the corresponding bins of the histograms from the four adjacent segments. As the final

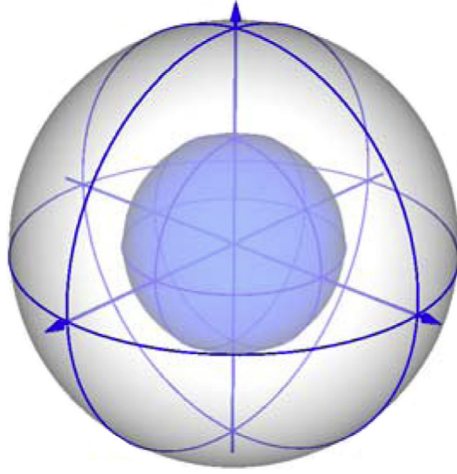


Figure 4.11: This figure from [73] shows the signature segmentation of the spherical neighborhood for a point $\mathbf{p}$ used by the SHOT descriptor algorithm. Histograms for all segments are combined to form the local 3-D point description.

step the whole description is normalized to have an Euclidean norm of 1, which is done to achieve robustness to point density variations.

According to the results of their experiments, they recommend histograms with 11 bins. Furthermore, they recommend a segmentation with 8 azimuth divisions, 2 elevation divisions, and 2 radial divisions (please note: Figure 4.11 shows 4 azimuth divisions). This leads to a description with total length of 352 values.

4.1.13 Unique Shape Context

Another local 3-D feature descriptor which was also introduced by Tombari et al. in 2010 is dubbed 'unique shape context' (USC) [92]. It is an extension of the 3-D shape context from Frome et al. summarized in Section 4.1.2.

The extension only consists of the adaptation of the local reference frame introduced by Tombari et al. within the context of the SHOT-algorithm (Section 4.1.12) to align the spherical volume to the local reference frame. This reduces the memory consumption, since, in comparison to 3-D shape context only one description per feature point must be stored in memory. In addition, the reduced number of feature descriptions reduces the ambiguity and therefore enables better detection and classification rates [4, 27].

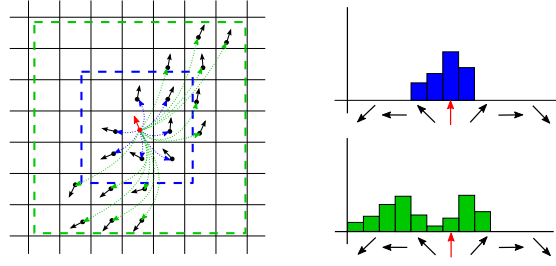


Figure 4.12: This figure from [20] outlines the computation of SURE descriptions exemplary for two dimensions. Discretized values of the Darboux frame are filled into histograms with 11 bins. The histograms for values of inner points (blue / upper histogram) and outer points (green / lower histogram) differ.

4.1.14 SURE

The algorithm introduced by Fiolka et al. in 2012 [20, 21] computes the feature description by means of the Darboux frame which was introduced in context of the point feature histogram by Rusu et al. in Section 4.1.5.

As already mentioned in the keypoint detection part of the SURE-algorithm (Section 3.1.12), Fiolka et al. determine points of interest for a neighborhood $\mathcal{N}_{\mathbf{p}, \mathcal{E}}$ based on a cubic region $\mathcal{E} \subseteq \mathbb{R}^3$. To compute the local feature description they use the same region and calculate the 4 values of the Darboux frame for each point pair $(\mathbf{p}, \mathbf{p}_i)$ with $\mathbf{p}_i \in \mathcal{N}_{\mathbf{p}, \mathcal{E}}$. They divide the range of values into 11 equidistant intervals and fill the values with respect to the corresponding interval into histograms with 11 bins. These histograms differ whether $\mathbf{p}_i$ is an inner point (i.e., if the point was within the radius used to approximate the normal vector) or an outer point. For two dimensions this is shown exemplary in Figure 4.12.

The final description is a concatenation of all 8 histograms, and accordingly has 88 dimensions. Furthermore, Fiolka et al. showed that the descriptor outperforms the matching score of the NARF descriptor introduced in Section 4.1.11.

4.1.15 Histogram of Oriented Normal Vectors

The histogram of oriented normal vectors (HONV) proposed by Tang et al. [89] in 2013 is a local 3-D feature description algorithm to be applied to depth images.

It is clear from the title, that this algorithm requires an approximation of normal vectors to compute the description. For a given depth image let the

depth values at a 2-D position (x, y) be $d_{x,y}$. To approximate a normal vector $\mathbf{n}_{x,y}$ at this position, Tang et al. use the cross product of vectors on the two tangent planes:

$$(4.23) \quad \mathbf{n}_{x,y} = \mathbf{t}_x \times \mathbf{t}_y, \text{ where } \mathbf{t}_x = \frac{\partial}{\partial x} \begin{bmatrix} x \\ y \\ d_{x,y} \end{bmatrix}, \mathbf{t}_y = \frac{\partial}{\partial y} \begin{bmatrix} x \\ y \\ d_{x,y} \end{bmatrix}.$$

This leads to a finite difference approximation:

$$(4.24) \quad \mathbf{n}_{x,y} = \begin{bmatrix} -\frac{\partial d_{x,y}}{\partial x} \\ -\frac{\partial d_{x,y}}{\partial y} \\ 1 \end{bmatrix} \approx \begin{bmatrix} -\frac{1}{2} (d(x+1, y) - d(x-1, y)) \\ -\frac{1}{2} (d(x, y+1) - d(x, y-1)) \\ 1 \end{bmatrix}.$$

Tang et al. found that the use of spherical coordinates is better than the use of Cartesian coordinates. Therefore, they determine the values as follows:

$$(4.25) \quad \begin{aligned} \varphi &= \tan^{-1} \left(\frac{d(x+1, y) - d(x-1, y)}{d(x, y+1) - d(x, y-1)} \right), \\ \theta &= \tan^{-1} \left(\frac{1}{2} (d(x+1, y) - d(x-1, y) + d(x, y+1) - d(x, y-1)) \right)^{\frac{1}{2}} \end{aligned}$$

Finally, Tang et al. compute a histogram for a sliding detection window of 8×8 pixel by counting the quantized values of ϕ and θ in a 2-D histogram, normalize the histogram and rotate the cells by a circular shift along each direction, so that the mean orientation is represented by the first bin. The last step is necessary to make the description rotationally invariant.

Unfortunately, Tang et al. give no information what size should to be used for the histogram. However, it can be seen in the figures of their work (see Figure 4.13) that a size of 9×6 is used.

4.2 Performance Evaluation of Feature Descriptors

Over the last years, many of the proposed algorithms were evaluated, e. g., from Bustos et al. [9], Heider et al. [30], and Alexandre [4]. Moreover, the available methods were summarized in a recent survey paper by Guo et al. [27].

However, no specific recommendations for or against certain methods are formulated in the conclusions of these evaluations. Instead, Alexandre summarizes an important point as follows:

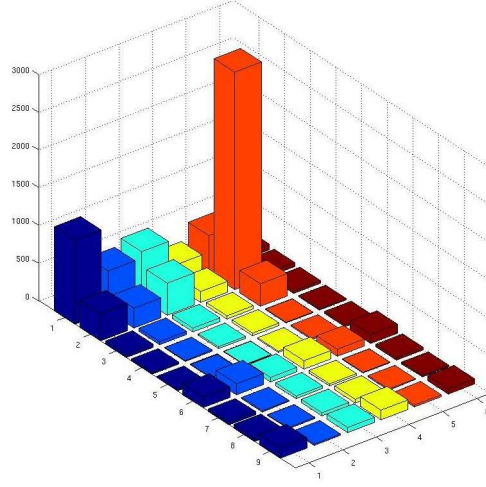


Figure 4.13: This histogram from [89] illustrates a final histogram of oriented normal vectors (HONV). The two dimensions represent the values of φ and θ corresponding to normal vectors.

“[...] since there are big differences in terms of recognition performance, size and time requirements, the descriptor should be matched to the desired task [...]” (Alexandre [4]).

4.3 Bag of Features

As already mentioned in Section 2.1, many of the classification pipelines use a bag of features model as a global description of a point cloud. The bag of features model has its origin in a field of computer science named natural language processing. There, a bag of words model is a histogram that counts the occurrences of words in a text without regarding the grammar. In the same way a bag of features is a histogram in which the occurrences of local feature descriptions are counted.

Since the number of possible feature descriptions is, in contrast to a natural language infinite, a defined subset of feature descriptions is required. Therefore, the feature descriptions need to be quantized to a finite number of representatives. This is often done with a k -means clustering.

Although invented by Lloyd in 1957 k -means was published accessible to the public only in 1982 [47]. The aim of k -means is to cluster the data in k partitions so that the sum of squared distances between each data element of a cluster and

the centroid of a cluster is at a minimum. In the bag of features context, the problem can be formulated as follows. Let $\mathcal{D}$ be a set of n -dimensional feature descriptions. Further, let $\mathcal{C}_i$ be one of the k clusters and $\mathcal{C}_1 \cup \dots \cup \mathcal{C}_k = \mathcal{D}$. Then the problem corresponds to the optimization of the function

$$(4.26) \quad \underset{\mathcal{C}_1 \cup \dots \cup \mathcal{C}_k = \mathcal{D}}{\operatorname{argmin}} \sum_{i=1}^k \sum_{\mathbf{d} \in \mathcal{C}_i} \|\mathbf{d} - \bar{\mathbf{d}}_i\|^2, \text{ where } \bar{\mathbf{d}}_i = \frac{1}{|\mathcal{C}_i|} \sum_{\mathbf{d} \in \mathcal{C}_i} \mathbf{d}.$$

The only parameter required in advance is the number of clusters k . Depending on the classification pipeline, the selected k differs by orders of magnitude. Toldo et al. use values of k between 20 and 80 [90] and values from 50 to 150 [91], Knopp et al. use 10% of all feature descriptions extracted from a training set as a value of k [41], Madry et al. use between 7 and 300 clusters [52, 51], and Yi et al. use 20% of the average number of features they extracted for each patch of all objects of their training set [103].

Given these significant differences, it is difficult to generalize the number of clusters. But there is another question in this context: is the Euclidean distance the best choice in distinguishing the feature descriptions while performing a k -means clustering?

The last question can partly be answered by a work of Madry et al. [51]. They compared, among others, different distance measures for a 33 dimensional fast point feature histogram (see Section 4.1.6). The evaluation is performed with two different test configurations. In the first configuration the original training objects were rotated and used as test objects. The average classification rates differ only slightly, i. e., a bag of features based on Jaccard distance for vectors [11] achieves $\approx 70\%$, while a bag of features based on Euclidean distance achieves $\approx 72\%$. In the second configuration Madry et al. use only objects where the test data differ significantly from the training set. This leads to an average classification rate of $\approx 36\%$ for the jaccard and of $\approx 46\%$ for the Euclidean distance.

Therefore, the Euclidean distance provides at least better results than the Jaccard distance for feature descriptions with a small number of dimensions (33 dimensions in this case).

5

Support Vector Machines

Object classification is the correct assignment of an object to one of a given set of abstract classes, e.g., cup, bottle, chair, and so on. In contrast to object recognition, any object, even previously unseen objects shall be classified correctly.

As already mentioned in Section 2.1, a common way to identify a class based on a given object description, i.e., a frequency histogram, is to use a classifier. Most of the current 3-D classification pipelines as well as the basic classification pipeline used in this thesis use a support vector machine. Hence, the support vector machine is described in more detail.

The support vector machine (SVM) introduced by Vapnik and Chervonenkis [96] is a learning algorithm that learns the relationship between a set of input vectors to an output. Initially, SVMs handled only problems with two categories. To show the principle of SVMs, Figure 5.1 depicts a simple initial example of data points associated with two disjoint classes.

5.1 Linear Separable Data

As shown, the data points in Figure 5.1 can be separated in 2-D by a straight line. For this simple example, the classification problem could be described as follows: find a straight line that separates the data points of the two classes maximizing the points' distances to the line. This approach corresponds, in essence, to the approach of an SVM: identify a linear separator – a straight line in this case – that separates the data points with a maximum margin.

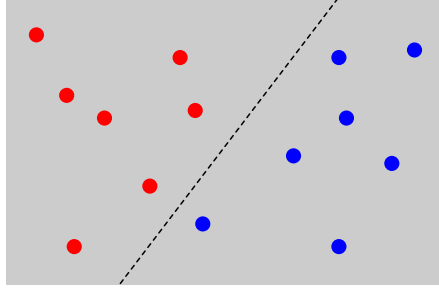


Figure 5.1: SVM example 1: linear separable data points in 2-D. associated with two disjoint classes.

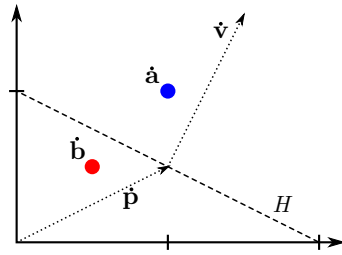


Figure 5.2: SVM example 2: two data points at $\mathbf{a} = (0.5, 0.5)^\top$ and $\mathbf{b} = (1.0, 1.0)^\top$ separated by a hyperplane H (dashed line) defined by a support vector $\mathbf{p} = (1.0, 0.5)^\top$ and a normal vector $\mathbf{n} = (0.5, 1.0)^\top$.

While Figure 5.1 shows the separation of the two categories by a straight line, a higher-dimensional space requires a hyperplane H . The location of the hyperplane in $\mathbb{R}^n$ is defined by a support vector $\mathbf{p} \in \mathbb{R}^n$ and a normal vector $\mathbf{v} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$:

$$(5.1) \quad H = \{\mathbf{x} \in \mathbb{R}^n \mid \langle \mathbf{v}, \mathbf{x} - \mathbf{p} \rangle = 0\}$$

Figure 5.2 shows an example with two data points in $\mathbb{R}^2$, where the hyperplane is defined by a support vector $\mathbf{p} = (1.0, 0.5)^\top$ and a normal vector $\mathbf{n} = (0.5, 1.0)^\top$. The figure also contains two points $\mathbf{a} = (0.5, 0.5)^\top$ and $\mathbf{b} = (1.0, 1.0)^\top$ lying on different sides of the hyperplane.

Whether, with respect to the direction of the normal vector, a point is above or below the hyperplane, can be determined with the plane equation. Then one has $\langle \mathbf{v}, \mathbf{a} - \mathbf{p} \rangle = -0.5$ and $\langle \mathbf{v}, \mathbf{b} - \mathbf{p} \rangle = +0.5$. These values correspond to the distance between the data point and the support vector $\mathbf{p}$. The sign reflects the side of the hyperplane. The different signs can be used for a classification. In connection with SVMs these labels are particularly specified as $y_i \in \{-1, 1\}$.

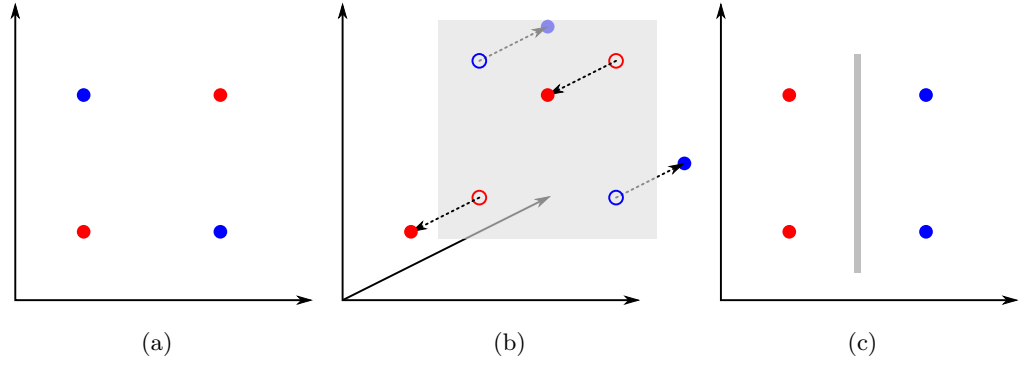


Figure 5.3: SVM example 3: (a) the XOR problem is a simple situation where a geometrical solution of the linear separation does not exist. In order to separate the data linearly (b), the original space must be left to rearrange the data points in a higher dimensional space (here $\mathbb{R}^3$). The orthogonal representation of (b) viewed from the side results in (c), where a geometrical solution of the linear separation exists.

In summary, training an SVM means to determine the support vector $\mathbf{p}$ and the normal vector $\mathbf{v}$, that the hyperplane separates the classes with a maximum margin.

5.2 Nonlinear Separation and the Kernel-Trick

However, in many cases it is not possible to perform a linear separation of the data into two classes. An often referred example is the XOR problem shown in Figure 5.3.

To get a linear separation of situations like the XOR problem, the original space must be left to rearrange the data points in a higher dimensional space (see Figure 5.3 (b)):

$$(5.2) \quad \phi : \mathbb{R}^n \rightarrow \mathbb{R}^m, \hat{\mathbf{x}} = \phi(\mathbf{x}), m > n.$$

Since it is very complex to transform a large amounts of data pairs into a higher-dimensional space and to search for an optimal hyperplane, a function is sought to map the hyperplane back from the higher dimensional space to the original space. This is known as the so called *kernel-trick*.

According to Mercer's theorem [55], the hyperplane of the linear separation in $\mathbb{R}^m$ can be transformed back to the original space $\mathbb{R}^n$, if the hyperplane has

a representation in the original space. This representation exists, if there is a continuous symmetric non-negative definite kernel function.

5.3 Kernel Functions

A continuous symmetric non-negative definite kernel function is given, for example, by the scalar product:

$$(5.3) \quad \mathcal{K}(\mathbf{x}, \mathbf{x}') = \langle \hat{\mathbf{x}}, \hat{\mathbf{x}}' \rangle = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle.$$

But this kernel function is primarily suitable for linear separable data and cannot solve the XOR problem [34] and therefore is also referred to as a linear kernel function. An extension to the linear kernel function is the polynomial kernel function:

$$(5.4) \quad \mathcal{K}(\mathbf{x}, \mathbf{x}') = (\langle \hat{\mathbf{x}}, \hat{\mathbf{x}}' \rangle + 1)^d.$$

For $d = 2$ the polynomial kernel function allows the solution of the XOR problem.

Another frequently used kernel function is the Gaussian kernel or the radial basis function kernel (RBF kernel):

$$(5.5) \quad \mathcal{K}(\mathbf{x}, \mathbf{x}') = \exp \left(-\gamma \|\hat{\mathbf{x}} - \hat{\mathbf{x}}'\|^2 \right), \text{ with } \gamma > 0.$$

Moreover, there are a variety of other kernel functions. But since the use of different kernel functions affects the SVM algorithm only marginally for linear separable cases, at this point no further kernel functions will be introduced.

Finally, it remains to be clarified how the kernel function is used as part of the SVM to determine the hyperplane. Beginning with the hyperplane equation $\langle \hat{\mathbf{v}}, \hat{\mathbf{x}} - \hat{\mathbf{p}} \rangle = 0$ a simplified version is given with $\hat{\mathbf{v}}^\top \hat{\mathbf{x}} - b = 0$, where $b = \hat{\mathbf{v}}^\top \hat{\mathbf{p}}$ [64]. As already mentioned above, the classes will be labeled with $y_i \in \{-1, +1\}$. It therefore follows that

$$(5.6) \quad \begin{aligned} \hat{\mathbf{v}}^\top \hat{\mathbf{x}} - b &\geq 0 \quad \forall \quad y_i = +1, \\ \hat{\mathbf{v}}^\top \hat{\mathbf{x}} - b &< 0 \quad \forall \quad y_i = -1. \end{aligned}$$

To take into account, that the margin on both sides of the hyperplane should be maximized, the margin takes a fixed size of 1, first:

$$(5.7) \quad \begin{aligned} \hat{\mathbf{v}}^\top \hat{\mathbf{x}} - b &\geq +1 \quad \forall \quad y_i = +1, \\ \hat{\mathbf{v}}^\top \hat{\mathbf{x}} - b &\leq -1 \quad \forall \quad y_i = -1. \end{aligned}$$

In combination with the class label this can be written as

$$(5.8) \quad y_i \left(\hat{\mathbf{v}}^\top \hat{\mathbf{x}} - b \right) \geq 1.$$

The absolute value of Equation 5.8 depends only on the length of the normal vector $\hat{\mathbf{v}}$. Thus, it is desirable to minimize the length of $\hat{\mathbf{v}}$ to maximize the margin between the data points. But an approach which takes only the normal vector length into account cannot keep the hyperplane between the two classes of data points. Instead, a solution with a hyperplane outside the data points and with infinitely wide margin would be found, since the length of the normal vector would tend to zero.

Therefore, a constraint should keep each data points on the right side of the hyperplane at a positive distance. This was already mentioned in Equation 5.8 and if this equation is considered as constraint during minimization, the hyperplane is kept between the classes. Therefore, the optimization problem can be summarized as follows:

- minimize $\hat{\mathbf{v}}^\top \hat{\mathbf{v}}$,
- for each data point $\hat{\mathbf{x}}$ let $y_i \left(\hat{\mathbf{v}}^\top \hat{\mathbf{x}} - b \right) \geq 1$.

The first item represents a quadratic function, which needs to be minimized, while the second item defines a linear constraint for each data point. This is a quadratic optimization problem which can be solved as follows [64]:

- Find positive Lagrange multipliers $\alpha_i \geq 0$, which let the following sum result in zero:

$$(5.9) \quad \sum_{i=0}^d \alpha_i \cdot y_i = 0,$$

where d is the number of data points to be considered.

- Maximize:

$$(5.10) \quad \sum_{i=1}^d \alpha_i - \frac{1}{2} \alpha_i \alpha_j y_i y_j \langle \hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j \rangle.$$

The last equation represents the linear case. Generally, the optimization of an SVM problem can be written as follows:

$$(5.11) \quad \sum_{i=1}^d \alpha_i - \frac{1}{2} \alpha_i \alpha_j y_i y_j \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j).$$

Based on Equation 5.11 it is possible to use different kernel functions for different classification scenarios. Once all parameters have been determined, the solution can be constructed as follows:

$$\begin{aligned} \hat{\mathbf{v}} &= \sum_{i=1}^d \alpha_i y_i \hat{\mathbf{x}}_i \\ (5.12) \quad &\text{and} \\ b &= y_i - \hat{\mathbf{v}}^\top \hat{\mathbf{x}}_i. \end{aligned}$$

6

Reinforcement Learning

Reinforcement learning is a machine learning method for sequential decision-making, in which through trial and error an optimal behavior at each stage of the sequence shall be learned. The goal is to learn the best possible way to solve a given task, which may consist of a plurality of different decision dimensions.

First of all a reinforcement learning system consists of an *agent* that interacts with an *environment*. Based on the current *state* of the environment the agent decides with respect to the learned experience what *action* will be performed next. The mentioned experience arises from consequences within the environment, i. e., a positive or negative *reward*, which reflects whether the action was appropriate to bring the agent closer to its *goal*.

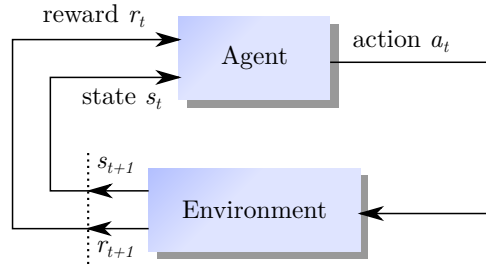


Figure 6.1: The agent-environment interaction in reinforcement learning, adopted from [83].

Figure 6.1 shows the main agent-environment interaction loop. At a discrete time step t the agent selects with respect to the current state s_t one of the available actions $a_t \in \mathcal{A}(s_t)$. When the action is performed the environment changes to state s_{t+1} . In addition to information about the new state s_{t+1} the agent receives a numerical reward r_{t+1} . Based on this reward the agent can learn to select the next and ideally best action. The goal of a reinforcement learning task is to find

a so called policy π that maximizes the long-term reward, i. e., to maximize the sum of future rewards.

6.1 Markov Decision Process

Before we consider how an appropriate policy can be determined using the rewards, the following few lines give a brief overview of how reinforcement problems and rewards are often modeled. Typically the issues of constructing or modeling a reward signal are not part of a reinforcement learning problem, but they help to enhance the understanding of reinforcement learning.

A *Markov decision process* (MDP) consists of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a set of rewards $\mathcal{R}_{ss'}^a$ received when action a leads to a transition from the state s to state s' , and a set of probabilities $\mathcal{P}_{ss'}^a$ that describes the probability that an action a in state s leads to a state s' .

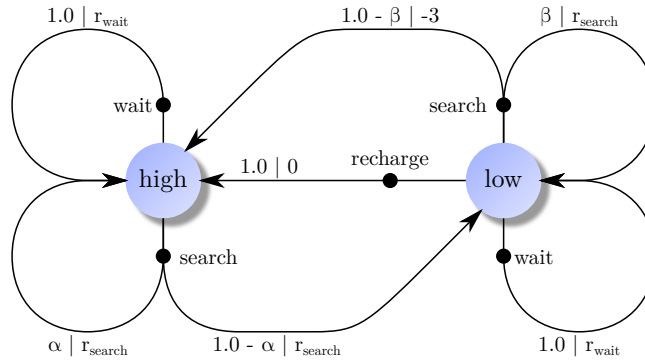


Figure 6.2: A simple Markov decision process with two states, three actions and various transitions, adopted from [83].

Figure 6.2 shows an exemplary Markov decision process of a recycling robot as introduced in [83]. The task of the robot is to collect empty soda cans in an office environment. The robot has two battery states, low and high. Based on these states the agent decides between three possible actions: searching for empty cans, waiting for someone to bring a can, and return to the base station to recharge the battery. In the latter case a recharge makes sense only when the battery level is low. Therefore, a recharging transition from state high to state high makes no sense. The wait action can be quickly explained, too. If the robot waits, it does not run down the battery power. Accordingly, the transitions for

this action keeps the state. Finally there are four transitions when the robot is actively searching for cans. If the battery state is high, it keeps high with a probability of α , and drops to low with a probability of $1 - \alpha$. When continuing the search of cans with a low battery state, it might work with a probability of β . But with a probability of $1 - \beta$ the battery is fully discharged. In the latter case the robot must be rescued and taken to the base station. This leads to an immediate reward of -3 . If $r_{research} > r_{wait}$, the robot will actively search for empty can, while it is worthwhile.

6.2 Reward and Return

As already mentioned above, a positive or negative reward reflects whether the action was appropriate to bring the agent closer to its goal. This means not the maximization of this immediate reward. Instead, the reward can be thought as a signal passing from the environment with the following informal idea:

“ That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal.” (Sutton and Barto [83])

This means, that the agent should maximize the cumulative reward in the long run. This cumulative reward is called *return*. Accordingly the sum of future rewards is called *expected return*. Thus, the goal is to find exactly that action which maximizes the *expected return*.

When the agent-environment interaction loop has a natural termination at time T or allows only a finite number of state transitions, the loop breaks into finite sequences of actions, so called *episodic tasks*. In this case the expected return can be formulated simply as

$$(6.1) \quad R_t = r_{t+1} + r_{t+2} + \dots + r_T.$$

In case of *continuing tasks* an additional concept called *discounting* is required and the value function has the following form:

$$(6.2) \quad R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where γ is the *discount rate* with $0 \leq \gamma \leq 1$. This function determines how strongly immediate rewards are weighted compared to rewards in the future. If

$\gamma < 1$ the distant rewards are less important and it is ensured, that the expected return is always a finite number if the immediate reward is bounded.

Subsequently, only the continuous case is considered, because the episodic case can easily be converted into a continuing task.

6.3 Value Functions and Policy

The *state-value function* estimates a value reflecting how good it is to be in a specific state. Essentially, the state-value function describes how much reward can be expected in a specific state. This value primarily depends on which actions will be performed in the future and which states will then be reached. This is done by the already mentioned policy π . A policy is a mapping of each state $s \in \mathcal{S}$ and each possible action $a \in \mathcal{A}(s)$ to a probability that action a is taken and applied in state s . Using this policy and using R_t the state-value function can be formalized as

$$(6.3) \quad V^\pi(s) = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\},$$

where $E_\pi\{\}$ denotes the expected value. Furthermore, a value function can be formalized where the value is separated for each available action in state s . This is called the *action-value function*, also known as *quality function* or simply *Q-function*:

$$(6.4) \quad Q^\pi(s, a) = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right\}.$$

6.4 Optimal Value Functions

For finite Markov decision processes it is possible to define an optimal policy. A policy π is better than a policy π' if the expected return for π is greater than the expected return for π' . This implies, that $V^\pi(s) \geq V^{\pi'}(s), \forall s \in \mathcal{S}$. This natural order of policies leads to a single *optimal policy* π^* . Hence, the optimal state-value function and action-value function will be defined as follows:

$$(6.5) \quad \begin{aligned} V^*(s) &= \max_{\pi} V^\pi(s), \\ Q^*(s, a) &= \max_{\pi} Q^\pi(s, a). \end{aligned}$$

An agent, that learns a policy that largely resembles the optimal policy is doing very well.

6.5 Learning Policies

Over the last years several types of reinforcement learning algorithms have been introduced. According to Konda and Tsitsiklis [42] they can be divided into three classes: actor-only, critic-only and actor-critic methods. Konda and Tsitsiklis use actor and critic as synonyms for policy and value function.

The following algorithm from Watkins and Dayan [100] is a critic-only algorithm called *Q-learning*. *Q*-Learning has some specific advantages over other algorithms [83]:

- It is a *model-free* technique, i. e., it can be used to compute optimal policies without a perfect model of the environment as a Markov decision process, in contrast to, for example, dynamic programming.
- It is proven to converge to an optimal value $Q^\pi(s, a)_t \rightarrow Q^*(s, a)$ with $t \rightarrow \infty$ [100].
- It learns iteratively building an optimal value function and is therefore computational efficient.

There are certainly a few restrictions on *Q*-Learning, but they should be of no consequence within the context in this work. These are (exemplarily):

- For large Markov decision processes the stored *Q*-table can become very large.
- Furthermore, large Markov decision processes have a problem that is known as the *curse of dimensionality*. This problem is expressed in the fact that a reinforcement learning method needs to visit each state-action pair sufficiently often to ensure convergence.

Introducing *Q*-learning, it is first assumed that the successor states for all $s \in \mathcal{S}$ are known. Then the optimal state-value function can be formulated as follows:

$$(6.6) \quad V^*(s_t) = r_{t+1} + \gamma V^*(s_{t+1}).$$

To approximate the optimal state-value function it is further assumed, that a fixed policy π is used:

$$(6.7) \quad V^\pi(s_t) = r_{t+1} + \gamma V^\pi(s_{t+1}).$$

```

Initialize  $Q(s, a)$  with arbitrary value (e.g., with zero)
repeat
  Initialize  $s$ 
  repeat
    Select action  $a$  from  $s$  using any policy (e.g., random or  $\epsilon$ -greedy)
    Perform action  $a$  and observe  $r$  and  $s'$ 
     $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ 
    Update the current state:  $s \leftarrow s'$ 
  until a terminal state is reached;
until no more episodes should be processed;

```

Algorithm 1: Q -learning algorithm

But the problem is, that $V^\pi(s_{t+1})$ is not known. Therefore, the current state value estimation $V_t(s_{t+1})$ is used instead. Furthermore, this estimation can be used to update older state values. Let s be the current and s' be the next state, as well as $V(s)$ and $V(s')$ be the values of the current and the next state. Then the update rule is

$$(6.8) \quad V(s) = V(s) + \alpha [r + \gamma V'(s) - V(s)],$$

where α is a parameter that controls the update rate. This equation is called *temporal-difference* estimation for V . There is only one snag with this solution: It is assumed, that the policy is fixed and thus a fixed action a is selected for each state s . This is called *on-policy* learning.

The important change that has led to the off-policy learning algorithm Q -learning, is the use of the action-value function instead of the state-value function. After observing a transition of the form $(s_t, a_t, r_{t+1}, s_{t+1})$ the value of the state-action pair (s_t, a_t) is estimated by the Q -function:

$$(6.9) \quad Q(s_t, a_t) = Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)].$$

In this way, the update of the Q -values $Q(s_t, a_t)$ is independent of the selected action, i. e., independent of the selected policy.

Algorithm 1 gives a brief overview of the entire Q -learning approach.

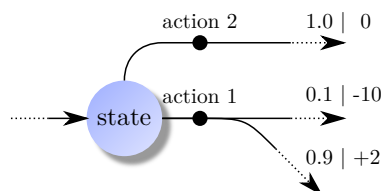


Figure 6.3: This illustrates a small part of an MDP.

6.6 Exploration-Exploitation Dilemma

In connection with the curse of dimensionality it was already mentioned that a reinforcement learning method needs to visit each state-action pair sufficiently often to ensure convergence. This means that in every state, all actions should be used as often as possible. This could be done systematically. However, since the values of different states and actions can relate to each other, a systematic approach can influence the results. Therefore, the actions are usually selected at random. Hence this will be called a *random policy*. Since this kind of *exploration* is needed due to uncertain action values, a random policy seems to be a preferable strategy to select actions.

However, if the current knowledge a reinforcement learning system has learned shall be applied to solve its goal, a random policy is inappropriate. Instead a policy that maximizes the expected reward in accordance with the objective of the reinforcement learning task is required. This can be achieved selecting the action with the greatest Q -value for each state. This is a greedy policy called *maxQ policy*. Selecting an action this way means *exploiting* the current knowledge.

But there are some reasons, why a pure exploiting may also be impractical:

- Let assume, that there is state as shown in Figure 6.3. The state allows two different actions. Action 1 leads to an immediate reward of +2 with probability 0.9 and to an immediate reward of -10 with a probability 0.1. This results in an average value of 0.8. The immediate reward for action 2 is always zero. It is assumed as well that the maximum Q -values of the subsequent states are relatively similar to each other. This means, that the Q -values of the two actions shown in Figure 6.3 depend only on the immediate reward. If, during exploration, action 1 results several times in that transition, which leads to a negative reward, the Q -value of action 1

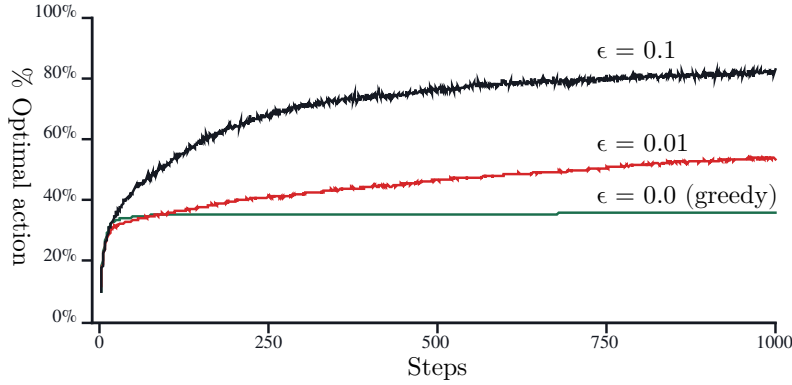


Figure 6.4: Exploration-exploitation dilemma with learning curvatures for greedy ($\epsilon = 0.0$) and ϵ -greedy policies with $\epsilon = 0.01$ and $\epsilon = 0.1$. Figure adopted from [83].

might be negative. In this case only action 2 would be used by a greedy policy, although action 1 would be the better choice in the long run.

- It must also be taken into account that in natural systems the environment and the results of actions may change over time. A reinforcement learning agent, that greedily takes only those actions corresponding to the best Q -value loses effectiveness over time.

In short, too much exploration prevents from maximizing the short-term reward (possible negative reward), but pure exploiting prevents from maximizing the long-term reward. This is known as the *exploration-exploitation dilemma* [83].

A simple way to solve this dilemma is to behave greedily most of the time. But with a probability of ϵ , the agent selects an action randomly. This method is called ϵ -greedy. Sutton and Barto use an 10-armed bandit problem to show the influence of different ϵ -values. Using $\epsilon = 0.1$ performs better than $\epsilon = 0.01$ or $\epsilon = 0$, but the ratio between Q^* and Q^π has more scatter (see Figure 6.4).

While using an ϵ -greedy approach, the actions will be selected indiscriminately with a probability ϵ . But there might be some actions with a higher potential for actually being optimal, i. e., actions with higher Q -values.

Let $N_t(a)$ be the number of how often action a was selected prior to time step t , then

$$(6.10) \quad Q_t(a) = \frac{R_1 + R_2 + \dots + R_{N_t(a)}}{N_t(a)}$$

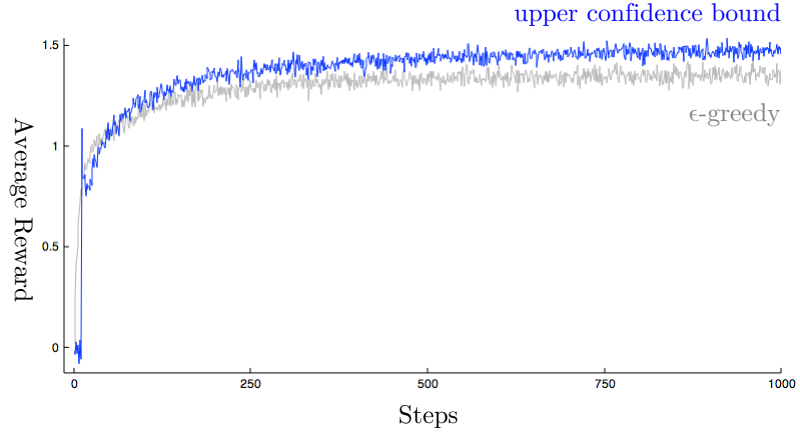


Figure 6.5: Upper confidence bound versus ϵ -greedy with $\epsilon = 0.1$ and $c = 2$ respectively. Figure adopted from [83].

is the mean reward received when action a was selected. Based on this, the actions can be selected as follows:

$$(6.11) \quad A_t = \operatorname{argmax}_a \left[Q_t(a) + c \sqrt{\frac{\ln t}{N_t(a)}} \right],$$

where c is a number that controls the degree of exploration.

This is called *upper confidence bound* (UCB). Figure 6.5 shows the difference between ϵ -greedy and the upper confidence bound using the 10-armed bandit. According to [83] this method seems to perform best.

This chapter can be seen as an overview over of the remaining chapters of this thesis. It briefly describes the problems when using local 3-D feature descriptors for classification tasks as well as the approaches that will be used to improve the classification results gradually.

This focus is on local 3-D feature descriptor algorithms for point clouds without additional structural information like triangle meshes and surfaces. These local 3-D feature description algorithms differ considerably with respect to descriptiveness and computation times. However, in general the computational costs of calculation and comparison are high.

7.1 Description of the Problem

Note that a recognition or classification task of objects which are represented exclusively as a 3-D point cloud of the surface of an object is a problem even for a human observer. This is particularly clear when one bears in mind how objects like apples, oranges or tennis balls look like without additional color information and without a continuous surface: they look equal.

A comparison of recent algorithms [4] and a survey of local feature based approaches for 3-D object recognition [27] show, that there is not a single best algorithm in the domain of 3-D object recognition. Concerning these two publications it must also be taken into account that they use local 3-D feature descriptors for object recognition and not for object classification tasks. That means, that they typically look at most for a hand full of objects which are usually very distinctive. Accordingly, it can be assumed that the classification rate of objects from large scale data sets with many similar object classes is much worse.

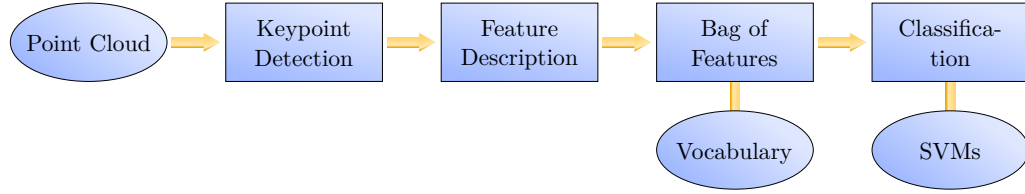


Figure 7.1: The structure of the baseline experiment to determine the classification rates of each feature descriptor.

This raises the question which descriptors should be used in which cases. Furthermore, the question arises whether a combination of two or more different algorithms leads to better results.

7.2 Approaches

The aim of this work is to show that a reinforcement learning system is able to increase the classification rate of 3-D objects by a skillful selection and combination of algorithms for local 3-D feature description.

7.2.1 The Basic Classification Pipeline

To ensure the comparability of the individual local feature descriptors, the classification rates of each individual local feature descriptor will be determined for different parameter settings of the basic classification pipeline. This involves a selection of a suitable keypoint algorithm, the determination of a proper bag of feature vocabulary size, and the identification of appropriate kernel parameters to train the support vector machines.

The structure of the baseline experiment is schematically shown in Figure 7.1 and is described in detail below.

Preprocessing

The preparation of 3-D point clouds is an important step toward a successful application of 3-D feature descriptors. Since the used point clouds may arise

from different sources, it is important to clean up the raw data first. This implies, for example, the elimination of point redundancies that may arise from the combination of several point clouds.

Thereupon the mesh resolution needs to be approximated based on the point cloud. This value is required for both the keypoint detection method, as well as for the local 3-D feature descriptor algorithms.

A full overview of the 3-D objects that are used to get the number of nearest neighbors which lead to the best approximation of the mesh resolution based on a point cloud can be found in the Appendix B.

Keypoint Detection

The keypoint detection algorithm used within all experiments has to be taken in advance. But considering the fact that there are still current pipelines that rely on sparse sampling (see Section 2.1), it was decided that sparse sampling will be initially included in the baseline experiments.

Secondly, a keypoint detection algorithm has been selected which was introduced by Zhong [105] in the context of intrinsic shape signatures (see Section 3.1.8). This algorithm is chosen, because both Salti et al. [72] as well as Filipe and Alexandre [19] (see Section 3.2) conclude that the intrinsic shape signature yields the best scores in terms of repeatability and is the fastest of the tested algorithms.

Local 3-D Feature Descriptors

When performing the basic classification pipeline, each of the 3-D feature description algorithms listed in Table 7.1 will be used separately to compute descriptions of the geometrical patterns around each of the keypoints extracted in the preceding step. A detailed description of all required parameters for each algorithm is given in Chapter 8.

The algorithms are chosen so that their properties are as heterogeneous as possible:

1. Three approaches use histograms (3DSC spherical / SI cylindrical / USC spherical), two approaches create a signature using surface properties (PFH

Table 7.1: This table gives a brief overview of the local 3-D feature description algorithm used in this work. The column 'Size' contains the size of the descriptions in bins of histograms or in dimensions of a corresponding feature vector. The third column 'Normal Vectors' indicates if normal vectors are required or computed, respectively.

Local 3-D Feature Description Algorithm	Size	Normal Vectors
Spin images (Johnson and Hebert [38])	153	yes
3-D shape context (Frome et al. [23])	1980	yes
Point feature histogram (Rusu et al. [66])	125	yes
Fast point feature histogram (Rusu et al. [71])	33	yes
Unique signatures of histograms (Tombari et al. [93])	352	yes
Unique shape context (Tombari et al. [92])	1960	yes

/ FPFH), and one algorithm is a hybrid solution of a spherical histogram and surface properties, i. e., normal vectors (SHOT).

2. The size of the feature descriptions should cover the largest possible area:
 $\text{FPFH} = 33 \rightarrow \text{3DSC} = 1980$.
3. In the same way the speed of the algorithms should cover a large area:
 $\text{SI} = 0.045\text{ms} \rightarrow \text{PFH} = 64.5\text{ms}$ (see Chapter 8).
4. Half of the algorithms require a local reference frame (SHOT / SI / USC)
half do not (3DSC / FPFH / PFH).

Vocabulary Construction

To combine the local 3-D feature descriptions in a global description, i. e., in a histogram, a finite vocabulary of quantized 3-D feature descriptions has to be determined. These quantized 3-D feature descriptions are often referred to as *visual words*. Each visual word is associated with a bin of the histogram, so that each local 3-D feature description can be assigned to its nearest visual word, represented by the corresponding bin.

This raises three issues, the first of which relates to the number of local 3-D feature descriptions that should be used to extract the visual words, while the second relates to the clustering algorithm and the distance measure, and the third

relates to the number of visual words of a bag of features vocabulary, i.e., the number of bins a histogram should have.

The answer to the first question is as follows: to show, that the determination of the visual words is to a certain extent independent of the objects used, only the first 10 ('apple' to 'cellphone') as well as the first 20 ('apple' to 'foodjar') of the 51 categories in alphabetical order from the RGB-D Object Dataset (see Section 7.3.1) will be used initially. To have a reference to compare to, the visual words will also be determined on the basis of all object categories.

In relation to the second question, Section 4.3 provides a commonly used combination: k -means clustering with an Euclidean distance. The initial centers of the clusters are chosen at random from the point cloud by using a variant proposed by Arthur and Vassilvitskii [5] in 2007, where they weigh positions of points according to the squared distance to the closest 3-D point already chosen. They show that their approach (named k -means++) leads to about 20% better results and is up to 70% faster in comparison with the standard k -means algorithm.

The third question can not be answered directly. The vocabulary should be large enough to represent relevant changes, but not so large as to distinguish between irrelevant variations caused by noise. The methods mentioned in Section 4.3 use values of k that differ by orders of magnitude, i.e., $k = 7$ versus $k = 300$. For this reason, the classification results for 7 different vocabulary and histogram sizes will be determined in this work:

$$(7.1) \quad k \in \{10, 20, 50, 100, 200, 500, 1000\}.$$

The best k will be used by the reinforcement learning framework.

Computation of Histograms

After the determination of the bag of features vocabulary, the computation of histograms is straightforward. It can safely be assumed, that the distance measure used to find the nearest visual word to each of the feature descriptions, should be the same as used for the k -means clustering: the Euclidean distance.

For each point cloud all visual words corresponding to the feature descriptions will be counted in bins of a so called frequency histogram.

Training a Classifier

As already stated in Section 5, most of the mentioned classification approaches use support vector machines as underlying technique. Rusu et al. [70, 65] state, that support vector machines have already been used for a classification based on frequency histograms of feature descriptions for color images with great success. In the referenced work, Rusu et al. test support vector machines, k -nearest neighbor searches and k -means clustering in different configurations against each other. The algorithm used in context of their classification is the point feature histogram.

The differences between the three mentioned methods are clearly visible. While k -means reaches only classification rates of $\approx 74\%$, the results of k -nearest neighbors are considerably better with classification rates up to $\approx 87\%$. But in comparison to the results that were achieved using a support vector machine, the results are rather weak, as can be seen in Table 7.2.

Table 7.2: This table shows an extract of the classification results from [70, Table 1]. The best results are achieved using a radial basis function (RBF).

Method	Result
SVM Linear kernel	95.17%
SVM Polynomial kernel	94.39%
SVM Sigmoid kernel	86.15%
SVM RBF Gaussian kernel	94.55%
SVM RBF Laplacian kernel	95.18%
SVM RBF Sublinear kernel	95.26%

In the work proposed by Lai et al. [45], two variants of support vector machines, strictly speaking, an SVM with a linear kernel and another SVM with a Gaussian radial basis function, and a random forest [7] are used. A random forest is a classification method, which consists of several different, uncorrelated decision trees. For a classification, each tree makes a decision and the class with the most votes will make the final classification.

Lai et al. also compare the results depending on whether they use only local 3-D feature descriptions, color image based features or a combination of both. Since only the classification rates of local 3-D feature descriptions are of interest in the context of this work, only the relevant results are included in Table 7.3.

Table 7.3: This table shows an extract of the classification results from [45, Figure 8]. The best results are achieved using a random forest. But the results of the support vector machine using a Gaussian radial basis function are close to this value. Considering the values shown in Table 7.2 the suspicion may arise, that a sublinear radial basis function could have led to an even smaller difference.

Method	Result
SVM Linear kernel	53.1%
SVM RBF Gaussian kernel	64.7%
Random forest	66.8%

The situation is similar in many other approaches. Johnson and Hebert [37], Kong et al. [43], Madry et al. [51, 14], and Yi et al. [103] use an SVM with a Gaussian radial basis function. Seib et al. [75] use an SVM without giving further information. Himmelsbach et al. [32] also use an SVM without specifying the kernel, but the given parameters indicate, that they use a radial basis function.

For this reason, an SVM with a Gaussian radial basis function will be used as a binary classifier for each object class.

Summary

In brief, the determination of best parameter settings used for 3-D object classification can be summarized as follows:

- Keypoint detection:
 - Keypoints determined with the intrinsic shape signatures keypoint algorithm
 - Keypoints based a sparse sampling
- Feature extraction:
 - Spin images
 - 3-D shape context
 - Point feature histogram
 - Fast point feature histogram
 - Unique signature of histograms
 - Unique shape context

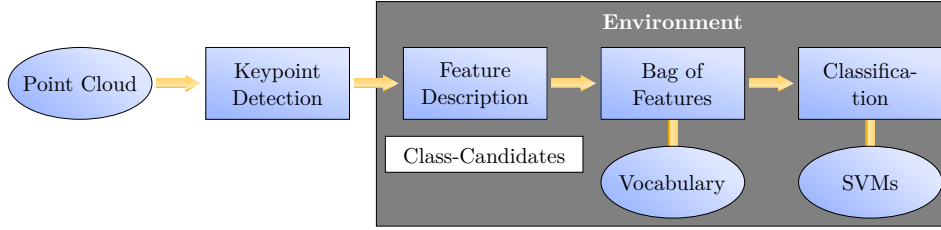


Figure 7.2: This illustration shows the basic classification pipeline as shown in Figure 7.1 embedded into the environment of the reinforcement learning model.

- Bag of features vocabulary:
 - Sizes: 10, 20, 50, 100, 200, 500, and 1000
 - Used object classes to extract the visual words:
 - 'apple' to 'cellphone', 'apple' to 'foodjar', and all object classes
- Bag of features histogram
- Classification:
 - Support vector machine
 - Kernel: Gaussian radial basis function

7.2.2 Fusion with Reinforcement Learning

Once the main parameters of the basic pipeline are determined, the pipeline will be successively fused with the reinforcement learning environment. To fuse the basic classification pipeline with a reinforcement learning framework, the pipeline is modified in a way, that a local 3-D feature description algorithms will be selected by a reinforcement learning agent, while all other parameters of the pipeline remain unchanged.

Initially, it is assumed that the environment of the proposed reinforcement learning framework is only defined on the class candidates. Hence, the computation of the feature descriptions, their conversion into a bag of features histogram and the classification are those components of the pipeline, which may lead to a change of the class candidates and of the environment, respectively. The keypoint detection is not taken into account, because the algorithm is given and fixed, and is only carried out once per input cloud. This is shown in Figure 7.2.

The next step is to extend the reinforcement learning model with the elements of the agent. The agent knows the current state of the environment, i.e., the

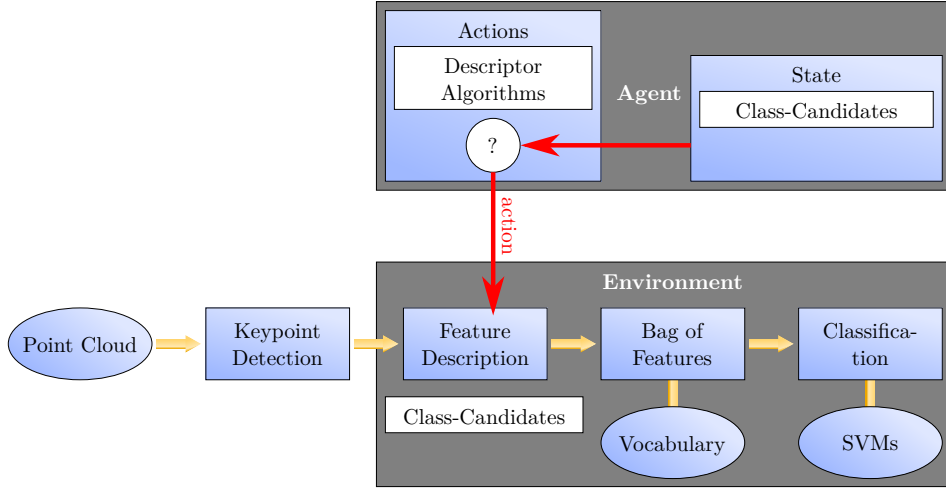


Figure 7.3: Extending the reinforcement learning model: the agent.

class candidates left. In addition, the agent has a list of actions. These actions correspond to local 3-D feature description algorithms the agent has not used at the time.

Based on the current state the agent can select the next action, i. e., the local 3-D feature algorithm that will be applied next. This is shown in Figure 7.3. At this point, the agent neither follows any policy or rule, nor learns anything, since there is no reward.

Therefore, it has to be clarified in which situations (states) the agent receives which amount of reward. For this purpose the states in which the reinforcement model should terminate must be defined in advance. For some of these cases, a special value is required, namely the sum that is calculated for each object class with the prediction values which have been determined for an object by the corresponding support vector machine while applying a sequence of local 3-D feature description algorithms.

Let us assume that c is an object class which is included in the remaining set of class candidates at step t : $c \in \mathcal{C}_t$. Further it is assumed that $\mathcal{S} = (a_1, \dots, a_t)$ is a sequence of local 3-D feature description algorithms a_i that have been applied on an object o within these t steps. Finally, $p_c(o, a)$ is a function, that returns the prediction value of the support vector machine for object class c when using algorithm a . Then the sum is defined as.

$$(7.2) \quad \sum_{i=1}^t p_c(o, a_i).$$

This sum, which is hereinafter referred to as *prediction sum*, describes the aggregated acceptance of an object by an object class over the application of several local 3-D feature description algorithms.

With the help of this short definition the terminal states can now be described:

1. If the set of class candidates does not contain the corresponding object class, a terminal state is reached and labeled as *fail state*.
2. If the computation time limit was exceeded two sub-cases can be distinguished. If the class candidate with the highest prediction sum
 - (a) corresponds to the correct object class, the state is counted as *overtime/match*.
 - (b) does not correspond to the correct object class, the state is counted as *overtime/miss*.
3. If the class candidates contains only a single object class, the state is counted as an *exact match*.
4. Finally, if no algorithm is left, another two sub-cases can be distinguished. If the class candidate with the highest prediction sum
 - (a) corresponds to the correct object class, the state is counted as *no actions/match*.
 - (b) does not correspond to the correct object class, the state is counted as *no actions/miss*.

If none of the above mentioned situations fits the current state, the reinforcement learning model has not reached a terminal state and thus the agent can select a new action. In the latter case, the selection of the value of the immediate reward is simple, because, since the selection of an action should not be influenced by external regulations, the reward is simply equal to zero. For all other cases the rewards are differentiated shown in Table 7.4.

As it can be seen in Table 7.4, the rewards for the terminal states are divided mainly into 4 groups, plus the reward for intermediate states of each episode. It starts with a negative reward of -1.0 for fail states, i. e., the situation in which it is impossible to get a correct result in any way. The neutral reward of 0.0 is used for intermediate states of an episode. Finally, three connected intervals

Table 7.4: Summary of rewards of the reinforcement learning model. Here is $\mathcal{C}$ the set of class candidates, $n_{\mathcal{C}}$ the number of all object classes, t the computation time required to get a result, and $t_{\max}$ the time limit.

case	name	reward	note
1	fail state	-1.0	the worst case
2(a)	overtime/match	$2.0 - \mathcal{C} /n_{\mathcal{C}}$	directly enables correct classification
2(b)	overtime/miss	$1.0 - \mathcal{C} /n_{\mathcal{C}}$	does not allow direct classification, but contains correct class
3	exact match	$3.0 - t/t_{\max}$	the best case
4(a)	no actions/match	$2.0 - \mathcal{C} /n_{\mathcal{C}}$	directly enables correct classification
4(b)	no actions/miss	$1.0 - \mathcal{C} /n_{\mathcal{C}}$	does not allow direct classification, but contains correct class

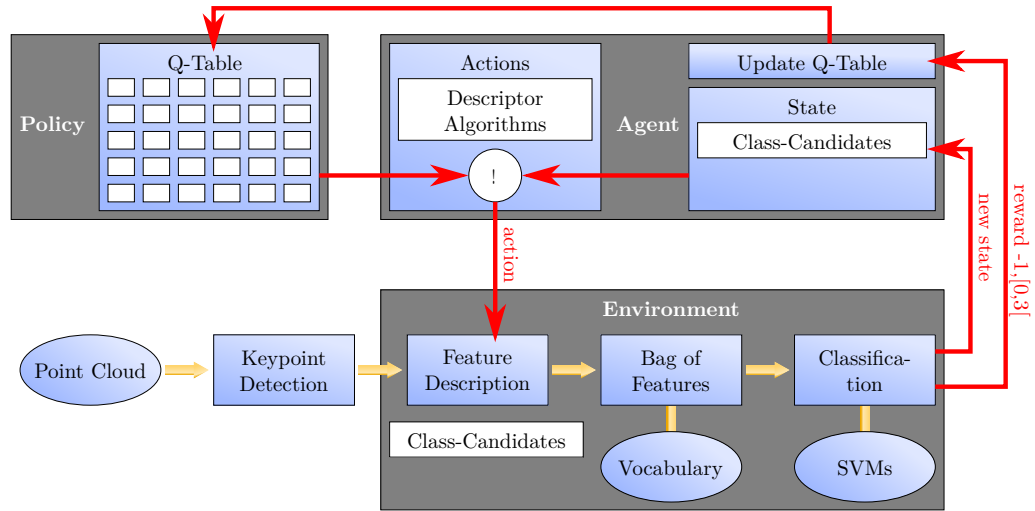


Figure 7.4: Extending the reinforcement learning model: reward and policy.

are used to describe the terminal states with more or less positive results. The cases 2(b) and 4(b), where the set of class candidates contains the correct object class, but where it is not possible to get a correct assignment, get rewards in an interval of $[0, 1[$ depending on the number of class candidates left. The cases 2(a) and 4(a), where the set of class candidates contains the correct object class and the class can be correctly assigned by the prediction sum lead to rewards in an interval of $[1, 2[$, which also depends on the number of class candidates left. Finally, the reward of an exact match is in an interval of $[2, 3[$ and depends on the computation time.

Using these specifications, which are summarized in Table 7.4, the reinforcement learning model can be extended and realized as shown in Figure 7.4. On the one hand this involves an approximation of the Q -values, i.e., learning the Q -table and on the other hand the ability to use a policy to select the next action. To what extent the policy uses the Q -table, of course, depends on the policy.

7.2.3 Differentiation of the First State

On closer inspection, it turns out that the previously presented model is neither adaptive nor uses the characteristics of the point cloud to influence the selection of the local 3-D feature description algorithm. The latter is the case, because the first state will always be the same, since the state is only given by the set of class candidates and it consists always of all 51 objects classes.

To overcome this, the state is enhanced by additional properties. These properties consist of characteristics of the point cloud. These will be:

1. The number of keypoints.

Note: one could also use the number of 3-D points of the point cloud. But this value correlates roughly with the number of keypoints. And because the number of keypoints has a direct impact on the computation time, the number of keypoints plays a more important role.

2. The ratios between the two successive eigenvalues of the covariance matrix of the point clouds, i.e., $r_1 = e_1/e_2$ and $r_2 = e_2/e_3$, where $e_1 \leq e_2 \leq e_3$.

Note: this indicates, whether the point cloud has a rather uniformly, flat, or elongated shape.

Since each of the aforementioned properties has a continuous range of values, the ranges will be divided into different intervals.

1. The number of keypoints:

1st quartile	→ small number of keypoints	→ 1
2nd and 3rd quartile	→ medium number of keypoints	→ 2
4th quartile	→ large number of keypoints	→ 3

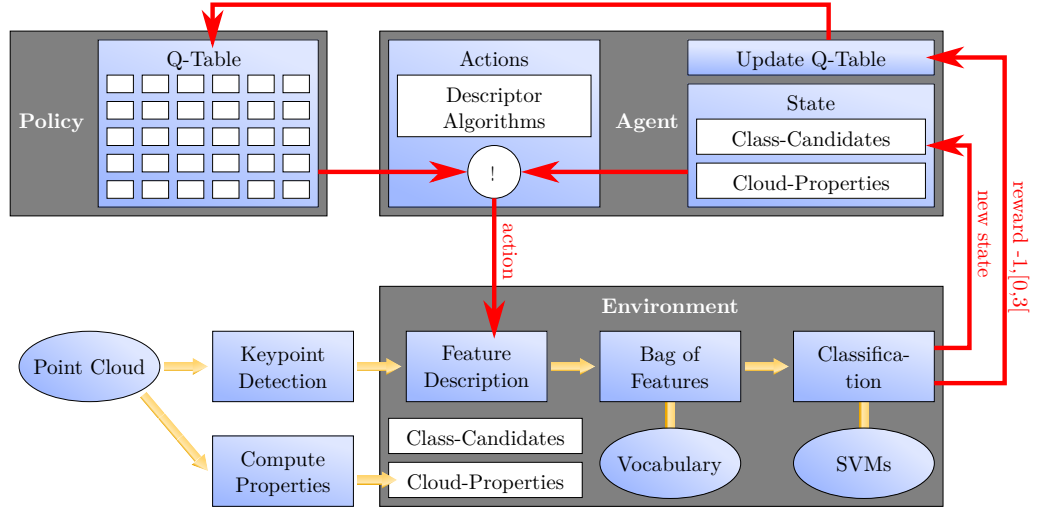


Figure 7.5: Extending the reinforcement learning model: point cloud properties.

2. The ratios between the two successive eigenvalues:

- | | | |
|------------------------------------|--|-----------------|
| $r_1 \leq 3.0 \wedge r_2 \leq 3.0$ | $\rightarrow$ in all dimensions rather uniformly | $\rightarrow 1$ |
| $r_1 \leq 3.0 \wedge r_2 > 3.0$ | $\rightarrow$ rather uniformly in two dimensions, but flat | $\rightarrow 2$ |
| $r_1 > 3.0 \wedge r_2 \leq 3.0$ | $\rightarrow$ rather elongated | $\rightarrow 3$ |
| $r_1 > 3.0 \wedge r_2 > 3.0$ | $\rightarrow$ elongated and flat | $\rightarrow 4$ |

Due to these different properties of a point cloud, the reinforcement learning agent is enabled to perform different actions in the initial state. This is shown in Figure 7.5 of the final reinforcement learning model.

7.2.4 Adaptive Learning

Furthermore, it must be clarified how the reinforcement learning model can respond to changes of the environment, i.e., to new local 3-D feature description algorithms.

This problem is directly related to a small restriction of the model:

The determination of immediate rewards is only possible when the object class is known in advance. Thus, initially a classification phase following a training phase with known objects can only use a max- Q policy. To be able to adapt the Q -table if a new local 3-D feature descriptor is added to the classification framework, the ϵ -greedy policy is adopted to ϵ -greedy episodes. This means that rather than sprinkling random actions with a probability of ϵ into each episode,

completely random episodes will be used with a probability of ϵ . During these random episodes only known objects will be used, so that the object class is known in advance.

7.3 Evaluation Methods

To ensure that the results of this thesis can be compared objectively with respect to the multitude of different algorithms, some requirements and restrictions are motivated hereafter.

7.3.1 Experimental Datasets

There are a lot of range image datasets available for different evaluation tasks. Many of them contain complete models, e. g., meshes of 5 objects obtained by a laser scanner [57]¹, meshes distributed over 5 datasets obtained from 29 individual objects and various synthetic scenes [93, 94]², 20 meshes of fully synthetic objects with scenes [63]³, or 35 CAD models in combination with real scenes containing the objects corresponding to the models [3]⁴.

Another dataset is the Berkeley 3-D Object Dataset (B3DO) [35]⁵, which contains depth images and RGB data of different scenes. Each scene contains rectangular areas labeled with one of 50 objects that can be found in each of these areas. However, the dataset does not contain an exact segmentation of the objects.

Thus, the previously mentioned datasets consist either of complete models, from which a variety of poses could be created, or of depth data, which is not sufficiently segmented. Another dataset that contains LiDAR and stereo depth data patches of 5 models as 3-D point clouds in various placements obtained by a laser scanner and with a stereo vision setup [88]⁶, is a step in the right direction. Nevertheless, there is still a problem that is related to all of the above mentioned datasets: within the context of object classification the datasets should contain

¹<http://www.csse.uwa.edu.au/~ajmal/recognition.html> – April 2015

²<http://www.vision.deis.unibo.it/research/78-cvlab/80-shot> – April 2015

³<http://www.dsi.unive.it/~rodola/data.html> – April 2015

⁴<http://users.acin.tuwien.ac.at/aaldoma/datasets/ECCV.zip> – April 2015

⁵<http://kinectdata.com/> – April 2015

⁶<http://rcvlab.ece.queensu.ca/~gridb/> – April 2015



Figure 7.6: RGB-D Object Dataset [45] from the Faculty of Computer Science and Engineering at the University of Washington. The dataset contains 300 objects in 51 categories provided as segmented point clouds and masked depth images.



Figure 7.7: Point clouds of Trimble 3D Warehouse [44] objects as part of the RGB-D Scenes Dataset v.2. The dataset contains segmented objects in 9 categories placed in different scenes to evaluate object recognition systems.

multiple objects of the same class, which is not the case for any of the datasets so far.

Finally, only two dataset were found that met all the criteria for a simple evaluation of classification tasks:

- The dataset should contain multiple labeled object classes,
- with multiple different objects within each object class
- and segmented patches of each object obtained from different locations, such that the size and the position of the objects varies.

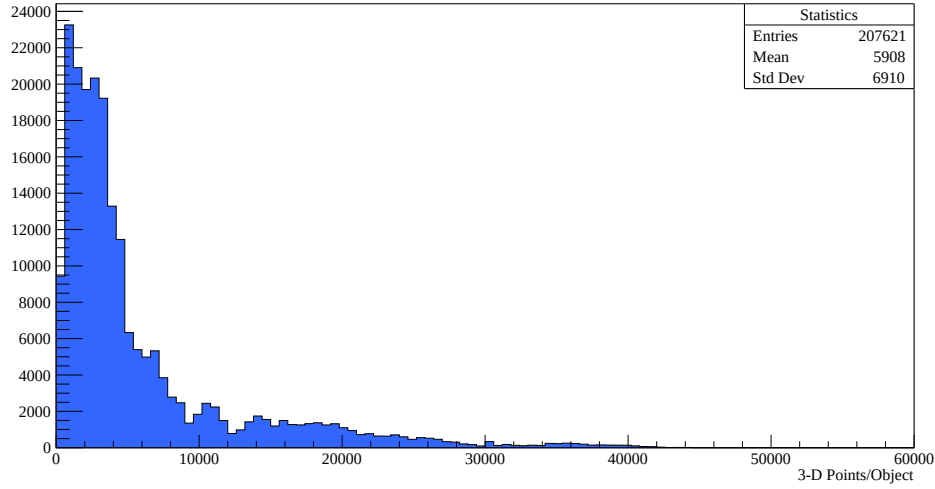


Figure 7.8: Distribution of point clouds of the RGB-D Object Dataset [45] according to the number of point.

These datasets are the RGB-D Object Dataset [45]⁷ shown in Figure 7.6 and the Trimble 3D Warehouse [44]⁸, shown in Figure 7.7. Both dataset come from the Faculty of Computer Science and Engineering at the University of Washington. Together they contain over 300 objects in more than 50 categories and all in all more then 200000 distinct patches of different poses. Thus, the two datasets are an ideal basis for the evaluation of the classification. Images of all 300 objects that can be found in these two datasets are shown in Appendix B.3. A detailed examination of the RGB-D Object Dataset is shown in Table 7.5.

Table 7.5: Properties of the RGB-D Object Dataset [45] from the Faculty of Computer Science and Engineering at the University of Washington.

Property	Value
Object classes	51
Distinct objects	300
Point clouds	207481
Mean points/cloud	≈ 5908.22
Median points/cloud	3296
Maximum points/cloud	59958
Mean point cloud resolution	≈ 0.001295

⁷<http://rgbd-dataset.cs.washington.edu/dataset/rgbd-dataset/> – April 2015

⁸<http://rgbd-dataset.cs.washington.edu/dataset/rgbd-scenes-v2/> – April 2015

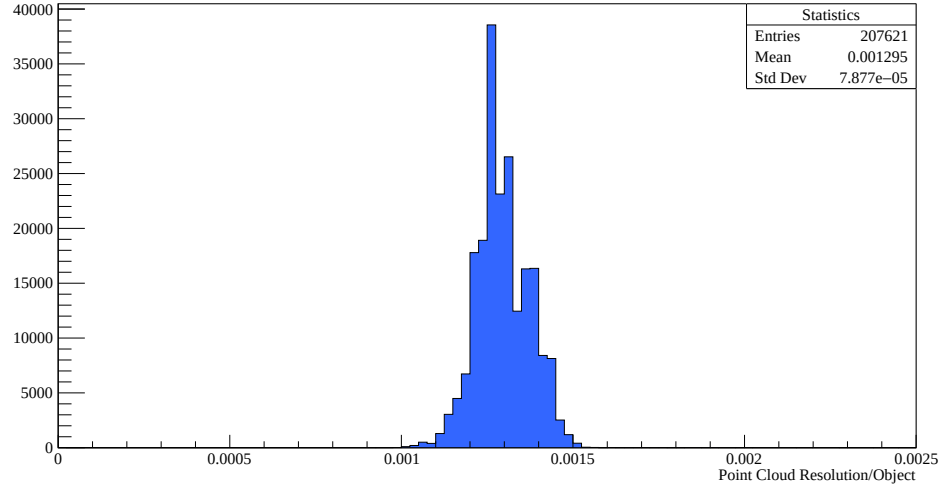


Figure 7.9: Distribution of point cloud resolutions of the RGB-D Object Dataset [45].

The distribution of sizes of all point clouds in this dataset is also visualized in the histogram in Figure 7.8. It is clearly recognizable that the majority of point clouds – exactly 140064 – lie in a range between 0 and 5000 points per point cloud.

A corresponding distribution of point cloud resolutions is shown in Figure 7.9. It can be seen that the densities of the point clouds vary only little in an interval $[0.0011, 0.0014]$.

7.3.2 Computation Times

The computation times of algorithms or parts of them will be determined at several different points. Since the computation times heavily depend on the system used, Table 7.6 gives a brief overview of the system used for all computations.

Table 7.6: System used for experiments.

Parameter	Value
System	Dell Precision WorkStation T3500
CPU	Intel Xeon E5630 @2.53GHz
Memory	12GB DDR3 @1066MHz
GPU	NVIDIA GeForce GTX 670
OS	Debian 8.0 (Jessie) GNU/Linux 64bit

A full overview of all hardware and software components, i. e., libraries, system software, development environment, and compilers can be found in the Appendix A.

7.3.3 Time Limit

For all subsequent experiments, in which the time limit does matter, the value for the time limit is set to 10 seconds. Table 7.7 anticipate the results of the required computation times of different local 3-D feature description algorithms shown in Chapter 8.

Table 7.7: Computation times of different algorithms. The second column is the average computation time of a feature description per keypoint in millisecond. The third column shows the average computation time for an object assuming an average number of 131 keypoints per object. These values anticipate results shown in Chapter 8.

Algorithm	t(ms)/keypoint	t(s)/object
3DSC	27.1	3.55
FPFH	6.69	0.88
PFH	64.5	8.45
SHOT	0.282	0.04
SI	0.0449	< 0.01
USC	9.95	1.30

The value of 10 seconds is chosen so that on the one hand it is usually unlikely, that all algorithms can be combined within this time limit, and on the other hand each algorithm can be applied individually without reaching the time limit.

7.3.4 Handling the Data

To be as efficient as possible, all classification results of known objects will be precomputed. Since the dataset consists of 51 classes it is possible to encode a classification result as bit pattern in a 64 bit long integer value. Furthermore, this enables fast bitwise logical operations.

Therefore, the classification results for each combination of object, object class and feature description are precomputed and stored in a long integer value. Within this long integer each object class has a fixed assignment to a single bit.

Additionally, the real computation times of the feature descriptors and prediction times as well as the responses and the class labels of the classification will be stored in a data structure. This enables a fast access to the classification results with the help of a lookup table to accelerate the learning and classification phase of the reinforcement learning environment.

8

Baseline Method

The results of applying different parameters at different stages of the standard classification pipeline will be discussed in this chapter. Just to keep the basic structure of this pipeline in mind, it is shown in Figure 8.1 once again.

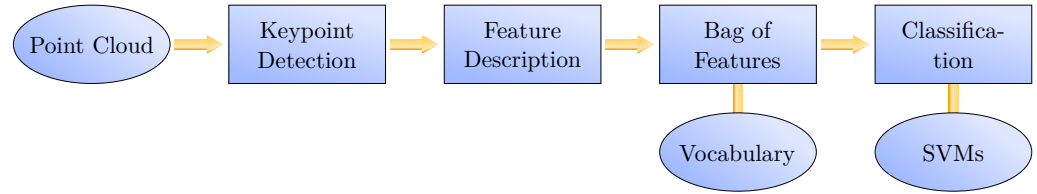


Figure 8.1: The structure of the baseline experiment to determine the classification rates of each feature descriptor.

8.1 Preprocessing

The preparation of 3-D point clouds is an important step toward a successful application of 3-D feature descriptors. Especially the point cloud resolution in terms of an approximated mesh resolution is a significant value, which is required by many of the subsequent algorithms. Therefore, the experiments start with a cleanup of the raw data and with the determination of the point cloud resolution to match a corresponding mesh resolution as good a possible.

8.1.1 Cleaning up Raw Data

The pipeline begins with a cleanup of the 3-D point clouds used as input for the classification pipeline. The remaining data structure consists only of an unsorted list of coordinates triples $(x, y, z) \in \mathbb{R}^3$.

To avoid potential problems with redundancies in point clouds (e. g., the nearest neighbor search will result in a point with the same coordinates as the reference point and thus with zero distance) it is important to remove redundant 3-D points. Due to the limited accuracy of 32-bit floating point systems, the redundancy is defined as follows: given a point cloud $\mathcal{P} \subset \mathbb{R}^3$, two points $\mathbf{p}, \mathbf{p}' \in \mathcal{P}$ are redundant, if the values of the three coordinates are approximately equal by a relative value of ϵ :

$$(8.1) \quad a \stackrel{\epsilon}{\approx} b \Leftrightarrow |a - b| < \max(a, b) \cdot \epsilon,$$

which leads to

$$(8.2) \quad \mathbf{p} \stackrel{\epsilon}{\approx} \mathbf{p}' \Leftrightarrow p_x \stackrel{\epsilon}{\approx} p'_x \wedge p_y \stackrel{\epsilon}{\approx} p'_y \wedge p_z \stackrel{\epsilon}{\approx} p'_z.$$

The ϵ used in context of this work is the difference between 1.0 and the next representable value: 0.000000119209.

Finally, all points of a point cloud will be sorted and compared with respect to the x -, y - and z -coordinate to remove the redundant points.

8.1.2 Approximation of the Point Cloud Resolution

Typically, the determination of the point cloud resolution is a straightforward task: compute the mean of the Euclidean distances between each point and its nearest neighbor. However, this approach is in several aspects not necessarily the best choice:

- Noisy data is not considered.
- Only the smallest distance of all direct neighbors is taken into account.

Especially the second point should, with respect to the sensors used to capture 3-D point cloud data, be considered more in detail. Regardless of whether the capture device is a time-of-flight camera, a rotating LiDAR system, a stereo vision system, a light section scanner, or a light coding system like the Microsoft Kinect, it is always a grid-like image of the depth values generated. This means, that any point – depending on the pixel connectivity one prefers – has 4 or 8 direct neighbors. On the other hand already processed point clouds are often structured as a triangle mesh. This leads to an average of 6 direct neighbors for each point.

Since most of the subsequently used algorithms use the *mesh resolution* as a measure for different properties, it is necessary to approximate the mesh resolution from unstructured point cloud data. The approximation of the mesh

resolution is done by an experiment. The datasets are obtained from *The Stanford 3-D Scanning Repository* [78] and from the *SHREC'13 - 3-D Shape Retrieval Contest 2013* [76]. The objects from the Stanford repository were captured using a Cyberware 3030 MS scanner ('Stanford Bunny', 'Happy Buddha', and 'Dragon') and the Stanford Large Statue Scanner ('Lucy'). With the exception of 'Lucy' all Stanford objects have 4 different resolutions, each. The SHREC'13 dataset consists of 10 numbered objects which were captured with the Microsoft Kinect camera. More details on the *The Stanford 3-D Scanning Repository* and the *SHREC'13 - 3-D Shape Retrieval Contest 2013* can be found in Appendix B.1 and B.2.

The mesh resolutions for all objects of Stanford 3-D Scanning Repository are shown in Table 8.1. They were calculated based on the Euclidean length of the set of unique edges.

Table 8.1: This tables show the mean edge lengths (*mesh resolutions*) of all test objects of the Stanford 3-D Scanning Repository.

object	mesh	object	mesh	object	mesh
nr.	resolution	nr.	resolution	nr.	resolution
Bunny	1	0.001471	Buddha	1	0.000345
	2	0.003049		2	0.000724
	3	0.006283		3	0.001515
	4	0.012605		4	0.003130
Dragon	1	0.000453	Dragon	1	0.000453
	2	0.000989		2	0.000989
	3	0.002052		3	0.002052
	4	0.004228		4	0.004228

The mesh resolution for Lucy is 0.487633. The mesh resolution values of all test objects of the SHREC'13 - 3-D Shape Retrieval Contest 2013 can also be found in Table 8.2.

Table 8.2: This tables show the mean edge lengths (*mesh resolutions*) of all test objects of the of the SHREC'13 - 3-D Shape Retrieval Contest 2013.

object	mesh	object	mesh	object	mesh
	resolution		resolution		resolution
22	1.610645	57	1.623456	146	1.714728
29	1.598942	141	1.624918	205	1.610277
49	1.619552	144	1.597935	210	1.632092
56	1.628885				

In addition, the mean Euclidean distances between the n nearest neighbors of m randomly selected 3-D points were calculated 20 times for each object, with $n \in [2, 10]$ and $m \in [1, 100]$. The differences of this mean values and the previously calculated mesh resolution were filled into a separate histogram for each value of n .

The histograms for different values of n , e.g., for values between 4 and 9 nearest neighbors of the point cloud from 'Happy Buddha' shown in Figure 8.2 are illustrative for all results. It shows, that 7 nearest neighbors lead to the best approximation of the mesh resolution of the corresponding triangle mesh.

Furthermore, it can be seen that the spread of approximated mean values at a sample size of 50 randomly selected points gets smaller. Since in the following experiments the local neighborhood of keypoint detection and feature description algorithms will have a size of at least 6 times the mesh resolution, these changes of $< 10\%$ of the mesh resolution (see the Table 8.3 for details) is small enough that the approximated mesh resolution can be used as a "stable" value for a point cloud resolution (pcr).

A complete overview is provided in Appendix C.1.

As already mentioned, Table 8.3 shows all results for a sample size of 50 random points. These values confirm once more, that most of the results with the smallest difference between the mesh resolution and the approximated values can be achieved using 7 nearest neighbors. For 'dragon1' and '210' 8 nearest neighbors provide the best results and for 'happy1' 9 nearest neighbors are the best.

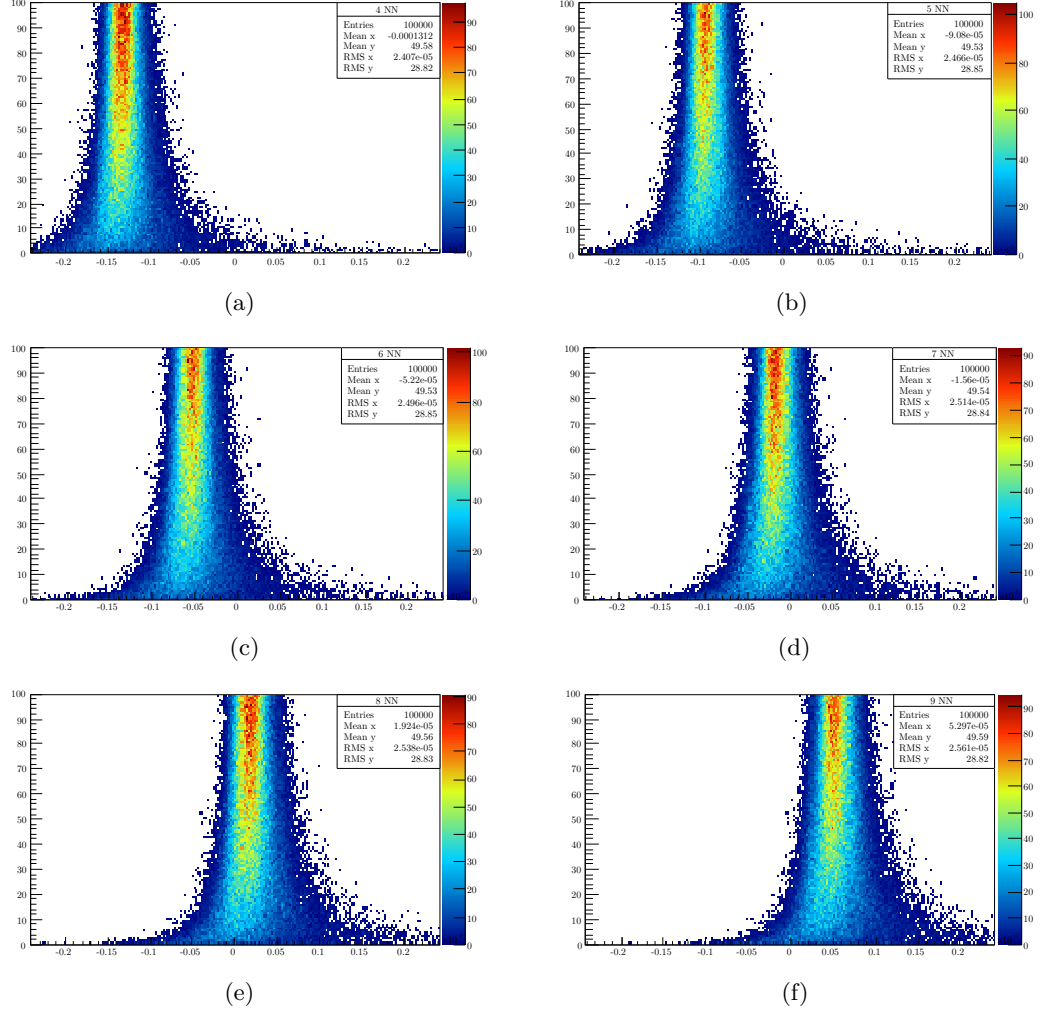


Figure 8.2: The distribution of differences between the mean point distances and the mesh resolution for 'Happy Buddha' of resolution 2. The x-axis shows the difference to the corresponding mesh resolution and the y-axis shows the number of randomly selected points which have been used to calculate the mean point distance to the n nearest neighbors.

Table 8.3: This table shows the best 3 results for approximation the point cloud resolutions with a sample size of 50 randomly selected 3-D points. The *n.n.* column show the number nearest neighbors used to calculate the point cloud resolution. The columns *app. pcr* and *mean diff.* show the approximation results of the point cloud resolution and the differences to the mesh resolution based on edge lengths. The column *diff.(%)* is the mean difference divided by the mesh resolution. The highlighted rows show the results with the smallest difference between the mesh resolution and the approximated point cloud resolution.

	n.n.	app. pcr	mean diff.	mean error	deviation	diff.(%)
bunny1	6	0.00143414	-0.00003648	± 0.00000057	0.00001813	-2.54
	7	0.00149832	0.00002769	± 0.00000059	0.00001860	1.85
	8	0.00155601	0.00008539	± 0.00000061	0.00001934	5.49
bunny2	6	0.00293929	-0.00010926	± 0.00000124	0.00003933	-3.72
	7	0.00308892	0.00004036	± 0.00000118	0.00003743	1.31
	8	0.00322700	0.00017844	± 0.00000114	0.00003604	5.53
bunny3	6	0.00601250	-0.00027031	± 0.00000243	0.00007673	-4.50
	7	0.00631754	0.00003474	± 0.00000232	0.00007326	0.55
	8	0.00660285	0.00032004	± 0.00000226	0.00007155	4.85
bunny4	6	0.01188754	-0.00071759	± 0.00000540	0.00017065	-6.04
	7	0.01248901	-0.00011612	± 0.00000526	0.00016623	-0.93
	8	0.01307578	0.00047065	± 0.00000505	0.00015977	3.60
dragon1	7	0.00042213	-0.00003129	± 0.00000057	0.00001813	-7.41
	8	0.00044624	-0.00000719	± 0.00000059	0.00001864	-1.61
	9	0.00046935	0.00001592	± 0.00000060	0.00001912	3.39
dragon2	6	0.00092701	-0.00006225	± 0.00000048	0.00001525	-6.72
	7	0.00097762	-0.00001164	± 0.00000048	0.00001513	-1.19
	8	0.00102583	0.00003656	± 0.00000048	0.00001518	3.56
dragon3	6	0.00194520	-0.00010637	± 0.00000082	0.00002589	-5.47
	7	0.00204684	-0.00000473	± 0.00000078	0.00002476	-0.23
	8	0.00214213	0.00009057	± 0.00000076	0.00002415	4.23
dragon4	6	0.00401105	-0.00021716	± 0.00000167	0.00005278	-5.41
	7	0.00421530	-0.00001291	± 0.00000160	0.00005074	-0.31
	8	0.00440744	0.00017923	± 0.00000156	0.00004927	4.07
happy1	8	0.00033166	-0.00001351	± 0.00000062	0.00001955	-4.07
	9	0.00034918	0.00000401	± 0.00000064	0.00002023	1.15
	10	0.00036598	0.00002080	± 0.00000066	0.00002091	5.68

Table 8.3 – continued from previous page

	n.n.	app. pcr	mean diff.	mean error	deviation	diff.(%)
happy2	6	0.00067256	−0.00005146	±0.00000052	0.00001650	−7.65
	7	0.00070928	−0.00001473	±0.00000054	0.00001698	−2.08
	8	0.00074425	0.00002023	±0.00000055	0.00001752	2.72
happy3	6	0.00143529	−0.00007986	±0.00000065	0.00002053	−5.56
	7	0.00150868	−0.00000646	±0.00000063	0.00002004	−0.43
	8	0.00157839	0.00006325	±0.00000063	0.00001987	4.01
happy4	6	0.00296176	−0.00016873	±0.00000124	0.00003915	−5.70
	7	0.00310749	−0.00002299	±0.00000122	0.00003844	−0.74
	8	0.00324530	0.00011482	±0.00000120	0.00003781	3.54
lucy1	6	0.45372069	−0.03391227	±0.00040262	0.01270632	−7.47
	7	0.47769997	−0.00993299	±0.00039771	0.01255136	−2.08
	8	0.50063982	0.01300686	±0.00036343	0.01146380	2.60
22	6	1.49075900	−0.105988623	±0.00097743	0.03090919	−8.04
	7	1.57352661	−0.03711863	±0.00090484	0.02861369	−2.36
	8	1.65140697	0.04076174	±0.00085199	0.02694224	2.47
29	6	1.48519538	−0.105374654	±0.00102160	0.03230577	−7.66
	7	1.56539965	−0.03354228	±0.00095677	0.03025567	−2.14
	8	1.64167482	0.04273289	±0.00090659	0.02866880	2.60
49	6	1.51214879	−0.10740342	±0.00106044	0.03353415	−7.10
	7	1.59634297	−0.02320924	±0.00101488	0.03209331	−1.45
	8	1.67625568	0.05670347	±0.00098079	0.03101521	3.38
56	6	1.51268232	−0.105620260	±0.00102263	0.03233830	−7.68
	7	1.59552272	−0.03336221	±0.00096106	0.03039137	−2.09
	8	1.67340803	0.04452311	±0.00091293	0.02886928	2.66
57	6	1.50576873	−0.105768710	±0.00099913	0.03159514	−7.82
	7	1.58644845	−0.03700738	±0.00094638	0.02992731	−2.33
	8	1.66332667	0.03987084	±0.00090344	0.02856938	2.40
141	6	1.50570396	−0.105921381	±0.00100312	0.03172154	−7.92
	7	1.59172876	−0.03318901	±0.00093044	0.02942320	−2.09
	8	1.67249761	0.04757984	±0.00088021	0.02783460	2.84
144	6	1.48245410	−0.105548125	±0.00105712	0.03342901	−7.79
	7	1.56492910	−0.03300625	±0.00100340	0.03173018	−2.11
	8	1.64288338	0.04494803	±0.00096563	0.03053598	2.74

Table 8.3 – continued from previous page

	n.n.	app. pcr	mean diff.	mean error	deviation	diff.(%)
146	6	1.59723832	-0.105748991	± 0.00117551	0.03717277	-7.36
	7	1.68091940	-0.03380884	± 0.00112022	0.03542443	-2.01
	8	1.75864098	0.04391275	± 0.00106408	0.03364913	2.50
205	6	1.49864524	-0.105163184	± 0.00100471	0.03177172	-7.45
	7	1.58350502	-0.02677207	± 0.00094614	0.02991970	-1.69
	8	1.66422949	0.05395240	± 0.00090713	0.02868590	3.24
210	7	1.59129154	-0.04080090	± 0.00104614	0.03308190	-2.56
	8	1.66709094	0.03499849	± 0.00099471	0.03145546	2.10
	9	1.74407467	0.11198222	± 0.00096867	0.03063199	6.42

At this point, it can be summarized, that the mean of the distances to the 7 nearest neighbors has been found as a suitable value for the point cloud resolution.

8.2 Keypoint Detection

As mentioned in Chapter 7 two different sets of keypoints will be used in comparison: keypoints based on sparse sampling and keypoints based on the intrinsic shape signature keypoint algorithm.

All relevant parameters for the intrinsic shape signature keypoint algorithm have been compared in the evaluation of Salti et al.. Thus, the decision is straightforward: While the repeatability rate of $\approx 75\%$ at a salient radius of 6 times the point cloud resolution (pcr) lies slightly below the absolute maximum of $\approx 80\%$ at a salient radius of $10 \cdot pcr$, the total number points decreased significantly from ≈ 180 at $6 \cdot pcr$ to ≈ 140 at $10 \cdot pcr$. Consequently, a salient radius of $6 \cdot pcr$ will be used for this algorithm.

To get a sense of the number of keypoints extracted when using the sparse sampling, the mentioned pipelines were examined for corresponding values. Johnson and Hebert [38] use 20% of all points as keypoints. Frome et al. [23] use a fixed number of 300 points. The objects that have been examined by Frome et al., contain an average of ≈ 60000 points, what would correspond to an selection rate of 0.5%. Drost et al. [17] select the samples with a distance of 0.05 times the diameter of the point cloud. With respect to a unit sphere this would correspond to a fixed number of ≈ 2900 points. Finally, Aldoma et al. [3] use two different

metric scales of $0.5cm$ and $1.0cm$ for point clouds of dishes. This results to a total number of $\approx 600/2400$ points for a cup of coffee with a height of $10cm$ and a diameter of $6cm$, and to a total number of $\approx 875/3500$ points for plate with a diameter of $26cm$. Both, Drost et al. and Aldoma et al., make no statements about how many points the point clouds actually contain. Thus, an estimation of the sparse sampling rate is not possible.

However, if the area covered by a keypoint is specified to have the same size as the intrinsic shape context keypoint, i.e., a salient radius of $6 \cdot pcr$, the corresponding circle would cover an area of $\approx 113 \cdot pcr^2$. Moreover, if it can be assumed that the areas covered by keypoints may not overlap, and if it can further be assumed, that the 3-D points of the point cloud are evenly spaced, one can approximate the number of keypoints, since the densest packing of circles in the plane is the hexagonal lattice [81]. This would lead to a subset of keypoints which is ≈ 1.25 of the original point cloud. But this is only a lower bound. If the point clouds – as it can be expected – have irregular spacings and holes, then the real number of keypoints is expected to be larger.

To summarize, two sets of keypoints will be computed and compared during the baseline experiments:

- Keypoints determined with the intrinsic shape signatures keypoint algorithm with a radius of $6 \cdot pcr$.
- Keypoints based a sparse sampling with a radius of $6 \cdot pcr$.

8.2.1 Number of Keypoints

It is primarily the number keypoints and the computation time that are of interest. Figure 8.3 contains the distributions of keypoints for the keypoint detection algorithm introduced with intrinsic shape signatures and sparse sampling.

As expected, the results of the sparse sampling look similar to the number of 3-D point of all objects shown in Figure 7.8. With an average of ≈ 355 keypoints, the number of keypoints is more than two and a half times higher, then the number of keypoints determined by the intrinsic shape signatures algorithm with an average of ≈ 132 keypoints per point cloud. If it turns out that the classification results using the keypoints from the intrinsic shape signatures algorithm are similar or even better than those with the keypoints determined by sparse sampling, the vastly reduced computation times of the feature descriptors

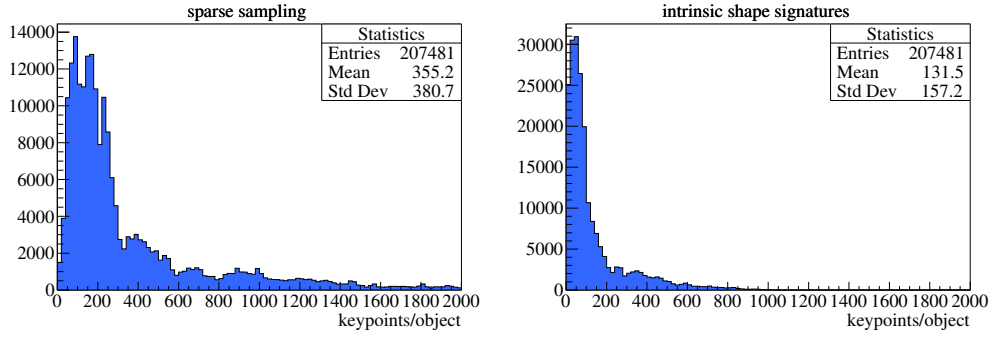


Figure 8.3: The distribution of keypoints per object. The histogram on the left side contains the number of keypoints for each of the 207481 objects determined by sparse sampling with a radius of $6 \cdot pcr$. Accordingly, the histogram on the right side contains the number of keypoints for each object determined by the keypoint detection algorithm introduced in context with the intrinsic shape signatures algorithm with the same radius.

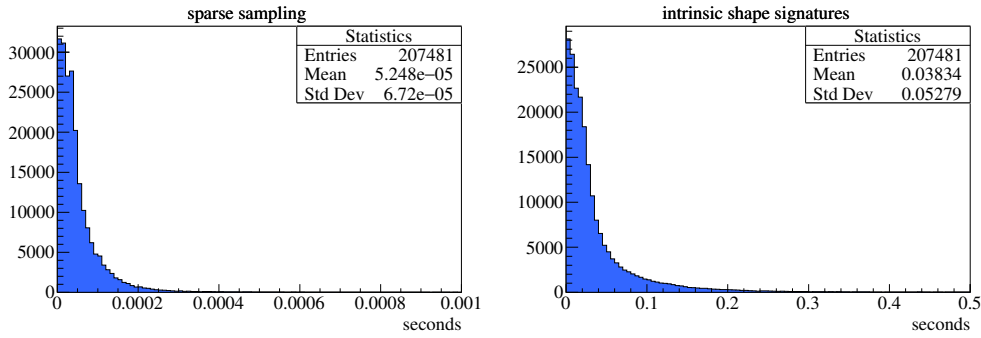


Figure 8.4: Computation times of a single keypoint. The histogram on the left shows the computation times of a single keypoint using sparse sampling. The right histogram shows the same using the intrinsic shape signature keypoint detection algorithm. Whilst at first glance the two histograms look very similar, the computation times using sparse sampling are more than 500 times smaller than those by the intrinsic shape signature.

and the bag of features histograms are worth the investment in a “real” keypoint detection algorithm.

8.2.2 Computation Time

When looking at the computation times for a single keypoint using the two keypoint detection approaches as shown in Figure 8.4, it quickly becomes clear that sparse sampling is magnitudes faster, i.e., more than 500 times than the intrinsic shape signature keypoint detection algorithm. While the mean computation

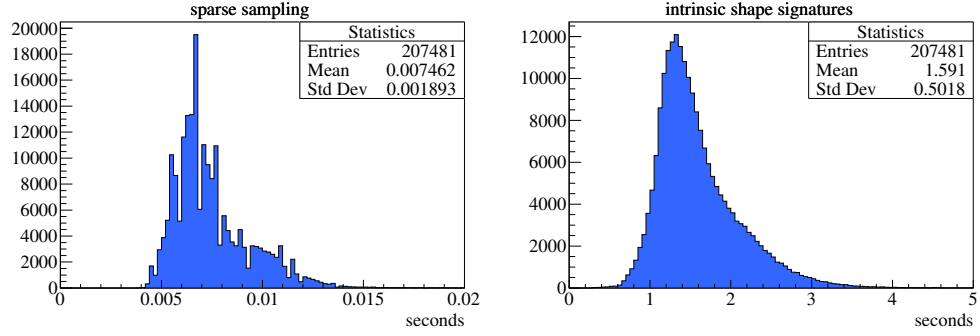


Figure 8.5: Computation times for the determination of keypoints per object. The left histogram shows the results for sparse sampling. The right histogram shows the corresponding values for the intrinsic shape signature keypoint detection algorithm.

time for sparse sampling is $\approx 0.05ms$, the computation of a keypoint using the intrinsic shape signature approach takes $\approx 38ms$ for a single keypoint.

Considering the mean number of keypoints per object, this would lead to mean computation times of $\approx 18ms$ and $\approx 5s$ respectively. But the results show, that the mean computation times per object have values of about $\approx 7.5ms$ for sparse sampling and of $\approx 1.6s$ for the intrinsic shape signature algorithm as shown in Figure 8.5. This significant difference suggests that there is no linear relation between the computation time and the size of an object, i. e., the number of keypoints.

8.2.3 Interim conclusion on Keypoints

The number of keypoints per object appears not to significantly affect the overall computation time for objects with a size similar to those used in this experiment. This relation is shown in Figure 8.6. It cannot be expected, however, that this holds for objects of any size. But for the objects of the RGB-D object dataset from the University of Washington the overall computation time seems to be dominated by a mostly constant overhead, e. g., method calls, instantiation of objects, or routines for memory allocation.

Therefore, at this stage of the experiments it can be stated, that the sparse sampling is magnitudes faster than the intrinsic shape signature keypoint detection algorithm. For the used objects the computation time for both can be assumed as nearly fixed: $\approx 7.5ms$ for sparse sampling and $\approx 1.6s$ for the in-

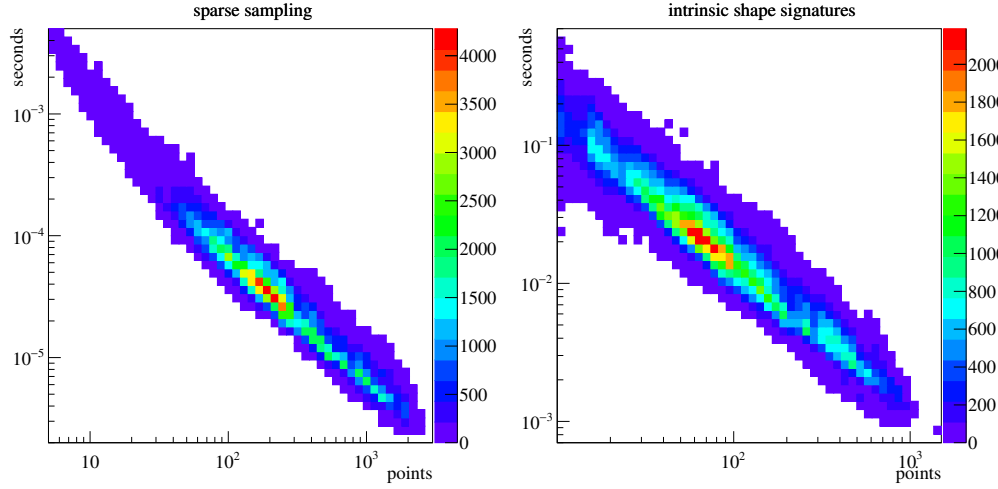


Figure 8.6: These two histograms show the computation time for a single keypoint in relation to the number of keypoint determined for the object. The histogram on the left shows the results for sparse sampling while the histogram on the right shows the same for the intrinsic shape signature keypoint detection algorithm. As can clearly be seen, a tenfold increase of the number of keypoints per object approximately leads to a tenth of the computation time for a single keypoint.

trinsic shape signature algorithm. It remains to be seen how much the keypoint detection contributes to the total computation time of the classification pipeline.

8.3 Local 3-D Feature Descriptors

The local 3-D feature descriptors used within this thesis require some parameters. Their values are taken from the publications of the respective algorithms where possible and subsequently discussed.

8.3.1 The algorithms

Spin Images

There are essentially three parameters to configure the spin image algorithm: the height and the width of the spin image, i. e., the number of bins, and the radius used for the determination of the normal vector. Since the original algorithm was designed for the use of meshes, the normal vectors are determined using the direct neighbor vertexes. The size of the spin image histograms used in the experiments

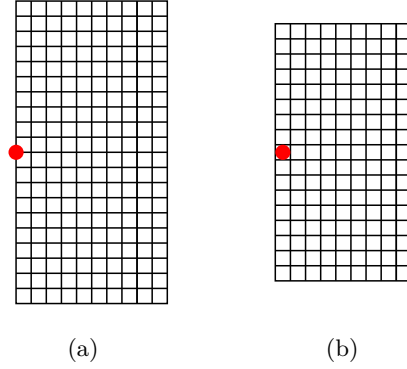


Figure 8.7: Two possible spin image placements, one with an even number of 20×10 bins (a) and one with an uneven number of 17×9 bins (b).

described in [38] was 20×10 . In a later work [37] Johnson and Hebert propose a spin image with a size of 15×15 , while Aldoma et al. [2] preferred a size of 17×9 . Johnson and Hebert place the keypoint vertically centered and horizontally aligned on the outer edge of the histogram, Aldoma et al. place the keypoint into the center of the bin in the middle of the first column (see Figure 8.7). The latter makes sense when using an uneven number of bins in both directions and a bin size identical with the point cloud resolution.

Since there are no further details regarding the determination of the normal vectors, they will be calculated based on the same points that will be used to compute the spin image. Accordingly, the radius for the determination of the normal vectors is $9 \cdot pcr$. The parameters for the computation of the spin images are finally summarized in Table 8.4.

Table 8.4: Summary of the spin image parameters.

Parameter	Value
Number of bins	$17 \times 9 = 153$
Radius to approximate the normal vector	$9 \cdot pcr$

3-D Shape Context

Frome et al. specify the values of all six relevant parameters within their publication. These are the area used to estimate the normal vector, the radius of the spherical region used to determine the description, and four other parameter,

that describe the segmentation of the sphere. However, all distance measures were specified in meters.

The used point clouds have an average point distance of $6cm$. The normal vectors were determined in a cubic area with a diameter of $55cm$, and $105cm$ for objects with larger noise. This is roughly equivalent to cubes with sides of $9.1 \cdot pcr$ and $17.5 \cdot pcr$, respectively. Spheres with the same volume would have radii of $5.6 \cdot pcr$ and $10.8 \cdot pcr$. According to their evaluation, the recognition rates are nearly identical with normal vectors based on these different radii. Therefore, the smaller radius seems to be sufficient and will be used in this work.

The support radius of the description is relatively large: $2.5m$, which corresponds to a radius of $\approx 41 \cdot pcr$. The description, i. e., the spherical histogram is divided into 15 radial divisions, 12 azimuth divisions, and 11 elevation divisions. The innermost sphere with a radius of $10cm$ ($\approx 1.6 \cdot pcr$) will not be considered.

Finally, Frome et al. require a radius to estimate the local point density. This radius is set to $20cm$, which is $\approx 3.3 \cdot pcr$. All parameters of the 3-D shape context are finally summarized in Table 8.5.

Table 8.5: Summary of the 3-D shape context parameters.

Parameter	Value
Radial divisions	15
Azimuth divisions	12
Elevation divisions	11
Number of bins	$15 \times 12 \times 11 = 1980$
Outer radius of the spherical histogram	$40 \cdot pcr$
Inner radius of the spherical histogram	$2 \cdot pcr$
Radius to approximate the normal vector	$6 \cdot pcr$
Density radius	$3 \cdot pcr$

Point Feature histogram

Rusu et al. also propose and compare the size of the k -neighborhood, on which the point feature histograms are calculated, in meters and centimeters with respect to two given scenes. The tested radii lie within an interval of $[2.0cm, 3.5cm]$ for an indoor kitchen scene and within $[50cm, 155cm]$ for an outdoor urban scene.

Rusu et al. assess the situation as follows:

“[...] As the point clouds are obtained directly from laser sensors by scanning the environment, the scale is known so the radii of a sphere can be chosen intuitively.[...]” (Rusu et al. [66]).

In addition, Rusu et al. do not provide any specific information on the determination of the normal vectors. Alexandre [4] gives an indication of the size of the area used for the approximation of the normal vectors: $1cm$. Since he uses the same dataset as it will be used in this work, the value corresponds to a radius of $\approx 7.7 \cdot pcr$ (the average point cloud resolution over all patches is 0.001295 – see Section 7.3.1).

Finally, there is the remaining question concerning the radius that will be used for calculating the feature descriptions. As mentioned by Rusu et al., the size of the description intuitively depends on the size of the object. The information given by Alexandre also does not help. He writes:

“[...] the values used were the ones set by default in PCL [...]” (Alexandre [4]).

The point cloud library (PCL) is a large scale, open project for 3-D point cloud processing [68]. Within the PCL the default value for the support radius of the point feature histogram is set to zero. For this reason it must be specified manually in each case. The only reference where a value is given, is a tutorial for the point feature histogram¹: 0.05 , which is a radius of $5cm$ or $\approx 38.6 \cdot pcr$ with respect to the test data used.

Since the test data mainly includes household objects and with an approximate diameter between $10cm$ and $30cm$, it is safe to assume that the seemingly small features have a maximum diameter of $5cm$ (radius $\approx 19.3 \cdot pcr$). This corresponds well to the radii that are used by Rusu et al. for the kitchen scene.

As described in Section 4.1.5 Rusu et al. prefer 5 subdivisions for each of the 3 values. Thus, the histogram has a fixed size of $5^3 = 125$ values. All parameters of the point feature histogram are summarized in Table 8.6.

¹http://pointclouds.org/documentation/tutorials/pfh_estimation.php – May 2015

Table 8.6: Summary of the point feature histogram parameters.

Parameter	Value
Number of bins used for this histogram	$5^3 = 125$
Radius of the spherical support area	$20 \cdot pcr$
Radius to approximate the normal vector	$8 \cdot pcr$

Fast point feature histogram

As the fast point feature histogram is based on the point feature histogram and follows the same mechanism, the radius of the spherical support area and the radius to approximate the normal vector will be the same as for the point feature histogram. However, the subdivisions of the values from the Darboux frame are structured differently: 11 bins per value concatenated to a description with 33 values.

All parameters of the fast point feature histogram are summarized in Table 8.7.

Table 8.7: Summary of the fast point feature histogram parameters.

Parameter	Value
Number of bins used for this histogram	$11 \cdot 3 = 33$
Radius of the spherical support area	$20 \cdot pcr$
Radius to approximate the normal vector	$8 \cdot pcr$

Unique signatures of histograms

Tombari et al. create histograms for a spherical environment, which they divided into several segments. They recommend histograms with 11 bins and a segmentation of the spherical environment with 8 azimuth divisions, 2 elevation divisions, and 2 radial divisions. This leads to 32 histograms and at least to a description with 352 values.

It is particularly pleasing that Tombari et al. specify an experimentally optimized size of the support area in point cloud resolution: $15 \cdot pcr$. Thus, all parameters given in their work can be found in Table 8.8.

Table 8.8: Summary of the unique signatures of histograms parameters.

Parameter	Value
Number of bins used for each histogram	11
Radial divisions	2
Azimuth divisions	8
Elevation divisions	2
Size of the description	$11 \times 2 \times 8 \times 2 = 352$
Radius of the spherical support area and to approximate the normal vector	$15 \cdot pcr$

Unique shape context

For the descriptor dubbed unique shape context, which is also proposed by Tombari et al., and which is an extension to the 3-D shape context, all parameters are given in detail. The parameters, which slightly differ in most of the parameters used for the 3-D shape context, are summarized Table 8.9.

At this point it should be noted, that Tombari et al. evaluate their descriptor in direct comparison to the 3-D shape context. But they do not use the parameters given by Frome et al.

Table 8.9: Summary of the unique shape context parameters.

Parameter	Value
Radial divisions	10
Azimuth divisions	14
Elevation divisions	14
Number of bins	$10 \times 14 \times 14 = 1960$
Outer radius of the spherical histogram	$20 \cdot pcr$
Inner radius of the spherical histogram	$2 \cdot pcr$
Radius to approximate the normal vector	$20 \cdot pcr$
Density radius	$2 \cdot pcr$

Now all local 3-D feature description algorithms used can be summarized as shown in Table 8.10.

Table 8.10: 3-D feature description algorithms used within the experiments.

Algorithm		Size
3-D shape context	(3DSC)	1980
Fast point feature histogram	(FPFH)	33
Point feature histogram	(PFH)	125
Signatures of histograms of orientation	(SHOT)	352
Spin images	(SI)	153
Unique shape context	(USC)	1960

8.3.2 Computation Times

In context of this work the computation times of the different local 3-D feature description algorithms are of special interest, since these values will be part of the decision process of the reinforcement learning framework.

The histograms shown in Figure 8.8 depict the distribution of computation times required by each of the algorithms to determine a single local 3-D feature description. The mean computation times differ in orders of magnitude.

Table 8.11: Computation times of 3-D feature description algorithms used within the experiments in ascending order. The last column shows the factor with respect to the fastest algorithm, the spin images.

Algorithm	Time	Factor
SI	$\approx 0.045ms$	1
SHOT	$\approx 0.28ms$	≈ 6
FPFH	$\approx 6.69ms$	≈ 150
USC	$\approx 9.95ms$	≈ 220
3DSC	$\approx 27.14ms$	≈ 600
PFH	$\approx 64.51ms$	≈ 1430

Table 8.11 shows the mean computation times and a factor that enables a quick comparison of the computation times with respect to the fastest algorithm, the spin images. So it must be expected that the reinforcement learning agent will start the classification with SI or SHOT if those two algorithms lead to similar or even better classification results than the other algorithms.

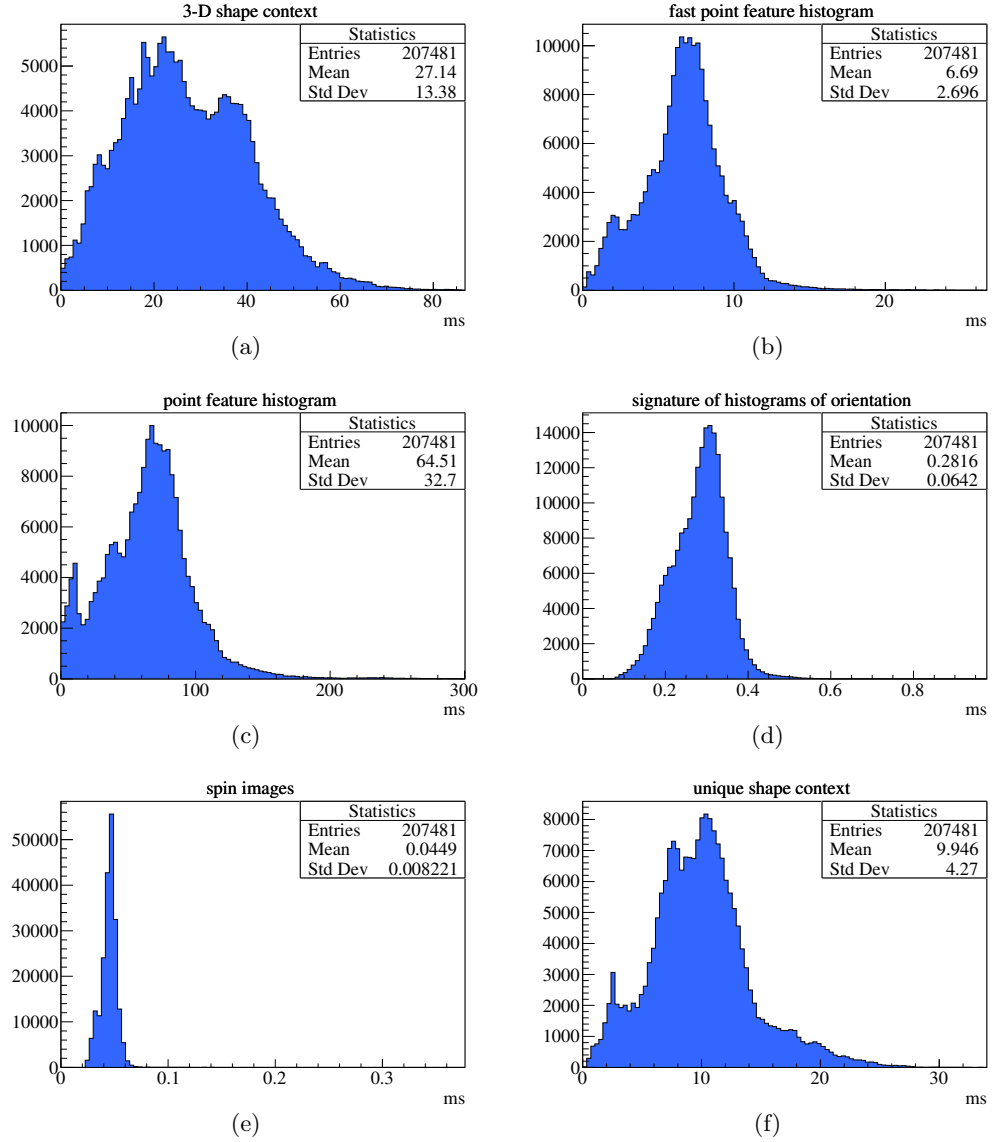


Figure 8.8: These six histograms show the computation times for a single feature description using (a) 3DSC, (b) FPFH, (c) PFH, (d) SHOT, (e) SI, and (f) USC. The computation times are measured in milliseconds. The difference between the smallest (SI with $\approx 0.045ms$) and the largest (PFH with $\approx 64.5ms$) computation time is tremendous (a factor of > 1400).

It should be noted that the results could be expected regarding some aspects. First, the PFH is an algorithm where many of the values will be redundantly computed multiple times. In addition the search of nearest neighbors in the local neighborhood, which is used intensively, is despite using data structures like a

k -d-tree a processing intensive task. It is therefore hardly surprising that this algorithm is so slow. The simplified variant of PFH, the FPFH, where most of the redundancy is removed, is therefore 10 times faster.

In case of 3DSC and USC the situation differs, since the algorithms share a simple structure. The reason for the large computation times is undoubtedly based on the large number of dimensions and the spherical structure of the histogram. The latter requires a computationally expensive calculation of angles. Without a doubt this can be done with a lookup table having about 2000 entries.

8.3.3 Interim conclusion on Local 3-D Feature Descriptors

Therefore, the following conclusion can be drawn regarding the used feature description algorithms: with regard to the significant differences in computation times, it will be interesting to see if and how the slow algorithms might be useful to identify a final object class from a probably already reduced set of class candidates.

8.4 Bag of Features

8.4.1 Vocabulary Construction

Concerning the baseline experiments, the vocabulary construction has no values or parameters which require an analysis. As described in Section 7.2.1 the computation of the vocabulary is done by a k -means++ clustering using the Euclidean distance. For each of the local 3-D feature description algorithms a set of vocabularies defined by all combinations of the following parameters is computed:

- 7 different vocabulary sizes (10, 20, 50, 100, 200, 500, and 1000)
- 2 different keypoint sets (sparse sampling / intrinsic shape signatures)
- 3 different subsets of the available object classes ('apple'-'cellphone', 'apple'-'foodjar', and 'all')

This leads to overall $6 \times 7 \times 2 \times 3 = 252$ vocabularies.

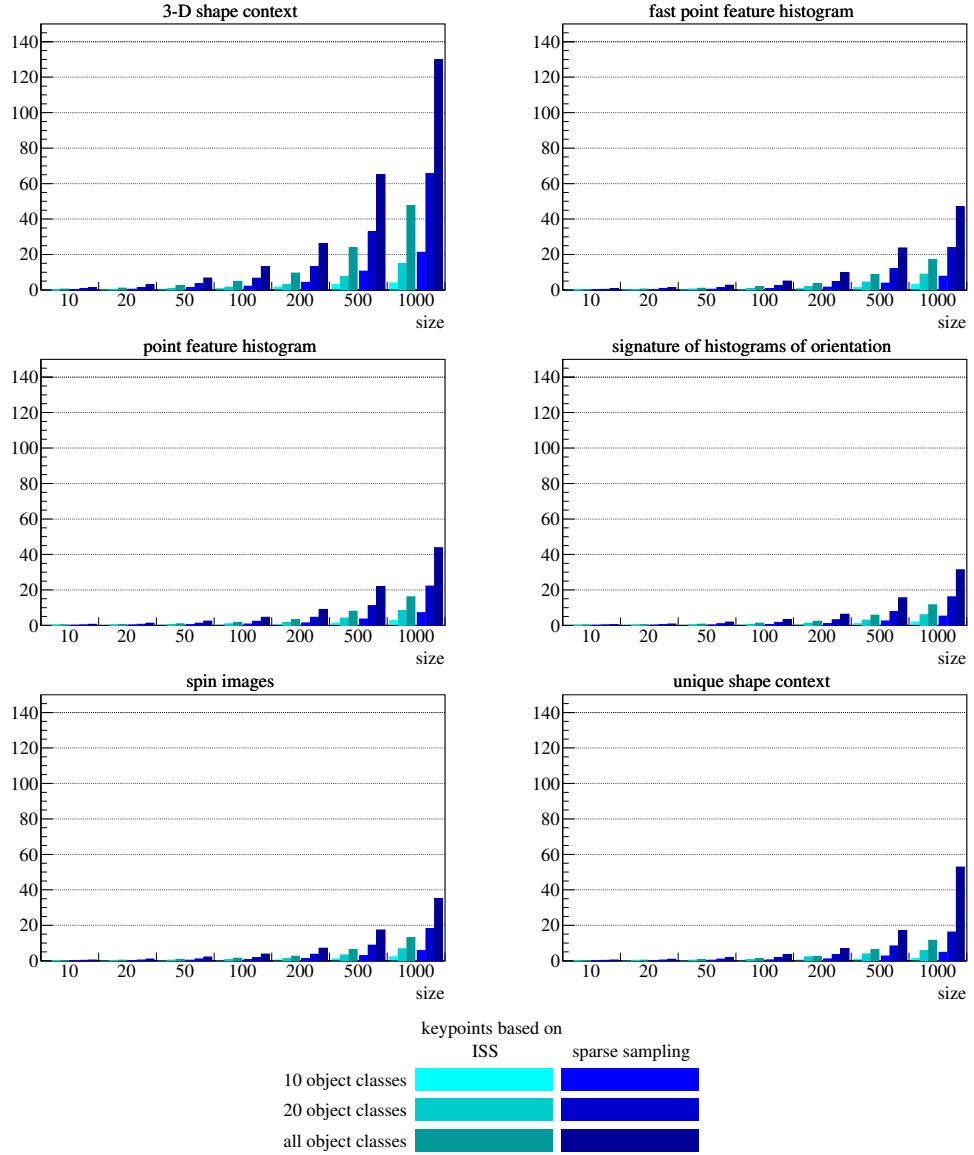


Figure 8.9: These graphs show the computation times for bag of features vocabularies in hours. The cyan colored bars correspond to feature vectors determined at keypoints based on the intrinsic shape signature algorithm. The blue bars correspond to feature vectors determined at keypoints based on sparse sampling.

Although the determination of the vocabularies is a one-time task and therefore has no relevance during classification, the computation times have been analyzed.

As Figure 8.9 illustrates the computation time rises with the number of object classes (10, 20, and 51) and with the mean number of features per object (≈ 130)

with ISS vs. ≈ 350 with sparse sampling) used during clustering. In other words, the computation time is approximately proportional to the number of features.

Surprisingly, with the exception of the 3-D shape context there is not much difference between the computation times of the different feature description algorithms. On the contrary, the computation time of SHOT with its 352 dimensions is less than the computation time of FPFH with only 33 dimensions. There might potentially be a correlation between the distinctiveness or descriptiveness of the feature descriptor and the computation time.

8.4.2 Computation of Histograms

The computation of histograms is a straight forward task without any intermediate results. For each point cloud, each keypoint algorithm, each local 3-D feature description algorithm, and each histogram size all visual words corresponding to the feature descriptions will be counted in bins.

This leads to 252 different histograms for each of the 207481 objects. This is an enormous number of 52285212 histograms that are hereinafter used to train the support vector machines.

8.5 Training and Classification Results

In this section the training and classification results will be analyzed. Within this context several aspects, i.e., the determination of optimal training parameters, the computation time of the training, the prediction time of a single classification, and the classification results will be discussed.

8.5.1 Optimal Training Parameters – a Single Case

As described in Section 7.2.1 a Gaussian radial basis function is used as the kernel function. The kernel function

$$(8.3) \quad K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}, \gamma > 0$$

has a parameter γ which has to be determined depending on the data which is used to train the support vector machine. Additionally, the support vector machine requires a parameter $C > 0$, which is the penalty parameter of the error

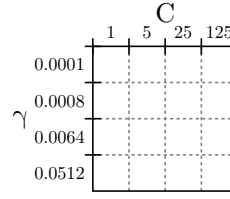


Figure 8.10: This figure shows the axes and labels of all histograms with parameters of the training phase of the support vector machines. Parameter C is the penalty parameter for misclassified samples and γ is the only parameter of the radial basis function.

term, i.e., a multiplier of the distance of misclassified samples to their region. This parameter must also be determined in combination with γ .

The following pages present a lot of small histograms with the size of 4×4 bins. All these histograms have the same axes and labels. To save space the axes and labels are not included for each histogram.

Instead, the labels and axes of all of these histograms are shown only once in Figure 8.10. The values of C increase from the left to the right, while the values of γ increase from the top to the bottom. The value ranges have been chosen due to preliminary experiments and cover the required ranges for all local 3-D feature descriptors. To reduce the computation time the ranges have been divided into 4 values using a logarithmic scale. This leads to a logarithmic step size of 5 for C and a logarithmic step size of 8 for γ .

In addition it makes no sense to discuss all details of the results separately. Accordingly, only details of some selected parameter combinations will be shown to illustrate the similarities and differences which have been caused by each of the parameters. The full set of results can be found in Appendix C.2.

All subsequent histograms match the following subset of test-parameters, until any of these parameters are explicitly changed:

- keypoint: ISS
- features: FPFH
- dictionary based on: 10 classes ('apple' to 'cellphone')

Figure 8.11 shows the classification results of a small subset of all pipeline parameters. The columns (a)-(g) are differentiated with respect to the 7 different sizes of the bag of features vocabularies (10, 20, 50, 100, 200, 500, and 1000). The histograms show the mean of the classification rates over all object classes. They illustrate, that the size of the bag of features vocabulary has only a minor

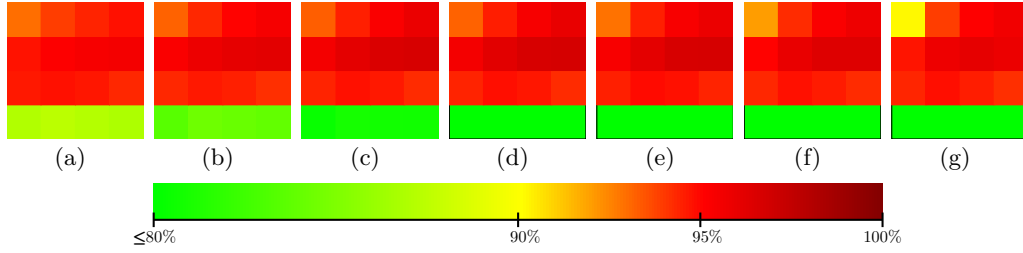


Figure 8.11: These figures show the mean classification rate for the local 3-D feature description algorithm FPFH with different values of the SVM parameters C and γ . The bag of features histograms used for training and classification were determined with ISS as the keypoint detection algorithm and with a vocabulary based on the first 10 object classes ('apple' to 'cellphone'). Each column of this figure corresponds to a bag of features vocabulary with a different number of entries: (a)-(g) = 10, 20, 50, 100, 200, 500, 1000.

affect on the classification rates for this parameter configuration. However, it is conspicuous, that there is a large difference of the classification rate when γ rises to a value of 0.0512. Then all classification rates drop to values of approximately 80% and below. To gain a better sense of the values, Table 8.12 shows the mean classification rates of Figure 8.12 (d), i. e., of a bag of features vocabulary with 100 entries.

Table 8.12: The mean classification rates of Figure 8.12 (d).

		C			
		1	5	25	125
γ	0.0001	93.10	94.46	95.33	95.90
	0.0008	95.46	96.18	96.57	96.65
	0.0064	94.33	94.73	94.59	94.19
	0.0512	79.35	79.88	79.77	79.72

It turns out that many parameter configurations reach the classification rates of $95\% \pm 1\%$ on average. It therefore does not make sense to select a support vector machine only on these values.

If one looks at the total number of object classes where a specific parameter combination leads to the best classification results, as shown in Figure 8.12, some parameter combinations will become more apparent. Table 8.13 shows the values of Figure 8.12 (d).

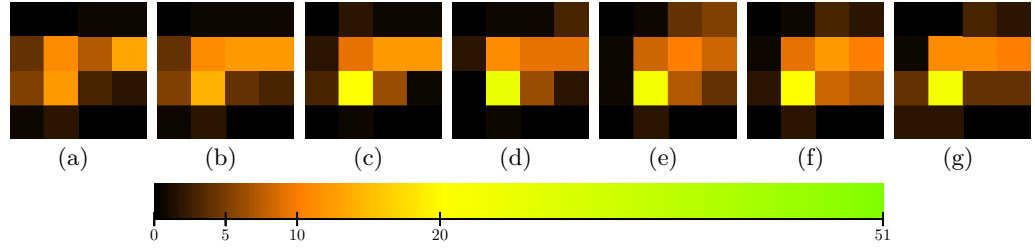


Figure 8.12: These figures show the number of object classes where a parameter pair (C, γ) lead to the best classification rate. The histograms are computed on the same classification pipeline parameters used for Figure 8.11. Analogously, each column of corresponds to a bag of features vocabulary with a different number of entries: (a) 10, (b) 20, (c) 50, (d) 100, (e) 200, (f) 500, (g) 1000.

Table 8.13: The number of best classification rates of Figure 8.12 (d).

	C			
	1	5	25	125
γ	0.0001	0	1	1
	0.0008	2	11	9
	0.0064	0	25	6
	0.0512	0	1	0

Here it becomes clear, that one would use the support vector machines with $\gamma = 0.0064$ and $C = 5$ to reach the highest recognition rates for a large portion of the object classes.

However, there is another aspect that should be envisaged in the context of this work: the computation time. On the one hand there is a prediction time, i.e., the computation time to achieve a decision if an input vector is within an object class or not, but there is also a computation time for the training phase of the support vector machine. Since the latter does not have an impact on the running classification process, the training times will only be shown to get a feel for how long a training will take, but not discussed in detail.

Figure 8.13 shows the training times of the subset of pipeline parameters used in this section. Significant differences with regard to the size of a bag of features vocabulary become apparent for the first time in the histograms shown in this figure. The larger the bag of features vocabularies, the higher the training times are. Apart from this, higher values of γ lead to significant longer training times. The training of the support vector machines of (g) with $C = 125$ and $\gamma = 0.0512$,

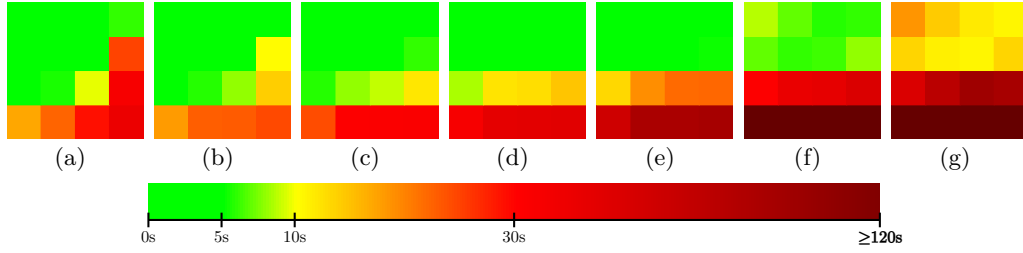


Figure 8.13: These figures show the training times for different parameter pairs (C, γ) . The histograms were determined under the same conditions as the figures shown previously, i. e., keypoints with ISS, features with FPF, and a bag of features vocabulary extracted from features of the first 10 object classes. Each column corresponds to a bag of features vocabulary with a different number of entries: (a) 10, (b) 20, (c) 50, (d) 100, (e) 200, (f) 500, (g) 1000.

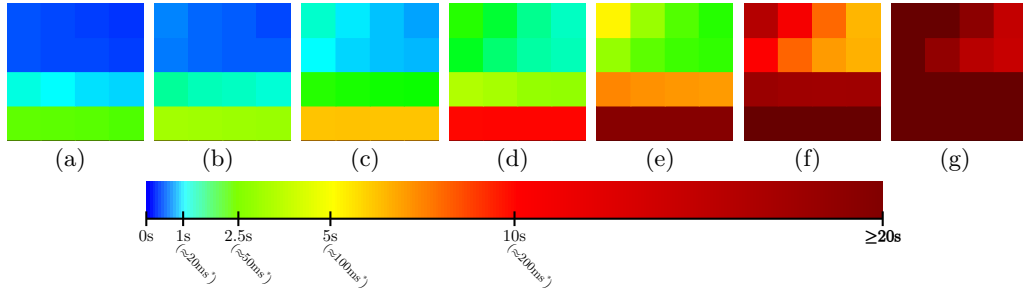


Figure 8.14: These figures show the prediction times for different parameter pairs (C, γ) . The histograms were determined under the same conditions as the figures shown previously, i. e., keypoints with ISS, features with FPF, and a bag of features vocabulary extracted from features of the first 10 object classes. Each column corresponds to a bag of features vocabulary with a different number of entries: (a) 10, (b) 20, (c) 50, (d) 100, (e) 200, (f) 500, (g) 1000. It is important to note that these values correspond to a prediction of ≈ 6000 input vectors: ≈ 2000 positive and ≈ 4000 negative examples. The values in parentheses with the superscripted asterisk correspond to a rough estimate for the required prediction time if for each of the 51 object classes a single prediction is performed.

for example, required ≈ 336 seconds on average and 859 seconds for the maximum case.

As already mentioned, the prediction times are of greater importance in context of this work. Thus, Figure 8.14 shows the prediction times for the pipeline setting as they have already been mentioned multiple times before. It is impor-

tant to note that these values correspond to a prediction of ≈ 6000 input vectors: ≈ 2000 positive and ≈ 4000 negative examples.

Later in this work, in context of the reinforcement learning framework, not a single support vector machine with a large set of input vectors but multiple support vector machines with exactly one input vector are invoked. Therefore, it will be presupposed that the prediction time of all support vector machines must not have a notable influence on the computation time at all. Thus, the computation time for a prediction of a single object class is limited to $2ms$. In case of the data used in this work, this will limit the prediction time of all 51 object classes to $\approx 100ms$.

Based on this limitation, the prediction times of Figure 8.14 can be interpreted as follows: when assuming an overhead of at least 50% for initialization and function calls, a single prediction should take at most $1ms$. Projected to the number of vectors used in Figure 8.14 this limitation is equivalent to $\approx 5s$. The values in parentheses with the superscripted asterisk in Figure 8.14 correspond to this rough estimate of the required prediction times.

Since especially those support vector machines are of interest, that are, besides being good classifiers, particularly fast, those support vector machines with an input space of 200 and more dimensions (Figure 8.14 (e)-(g)) can be excluded in this particular case of classification pipeline parameters.

8.5.2 Optimal Training Parameters – Comparison

All of the above mentioned histograms and values have been taken from a small subset of all (254) parameter combinations of the classification pipeline. How much do parameters influence the classification results at all?

The first parameter which is modified, is the number of object classes which have been used to determine the bag of features vocabularies. Figure 8.15 shows the results in comparison to Figure 8.11, which is equivalent to the first row of Figure 8.15 where the vocabulary is based on the first 10 object classes. The second and the third row of this figure show the same histograms with vocabularies based on the first 20 and all object classes, respectively. It can clearly be seen that there is absolutely no difference according to the recognition rates. Besides,

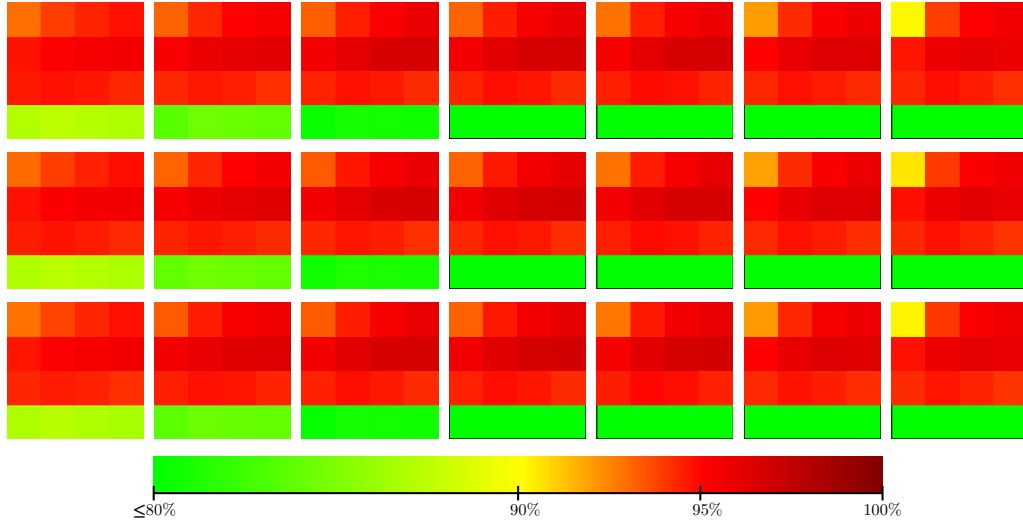


Figure 8.15: These histograms show the mean recognition rates of FPFH. The first row is equivalent to Figure 8.11. The histograms in the second row use vocabularies determined on 20 object classes, while the third row shows the same for vocabularies determined on all object classes.

the numerical values underline this result. Most of the mean classification results differ only in the second decimal place.

At this point a small intermediate result can be summarized:

If the number of local 3-D feature descriptions is large enough, the classification results do not depend on the subset of feature vectors that has been used to compute the bag of features vocabulary.

In the present case at least 5000000 feature vectors have been determined within the first 10 object classes using the keypoints from ISS. Thus, the smallest number of feature vectors that have been used for clustering the bag of features vocabulary was 5000000.

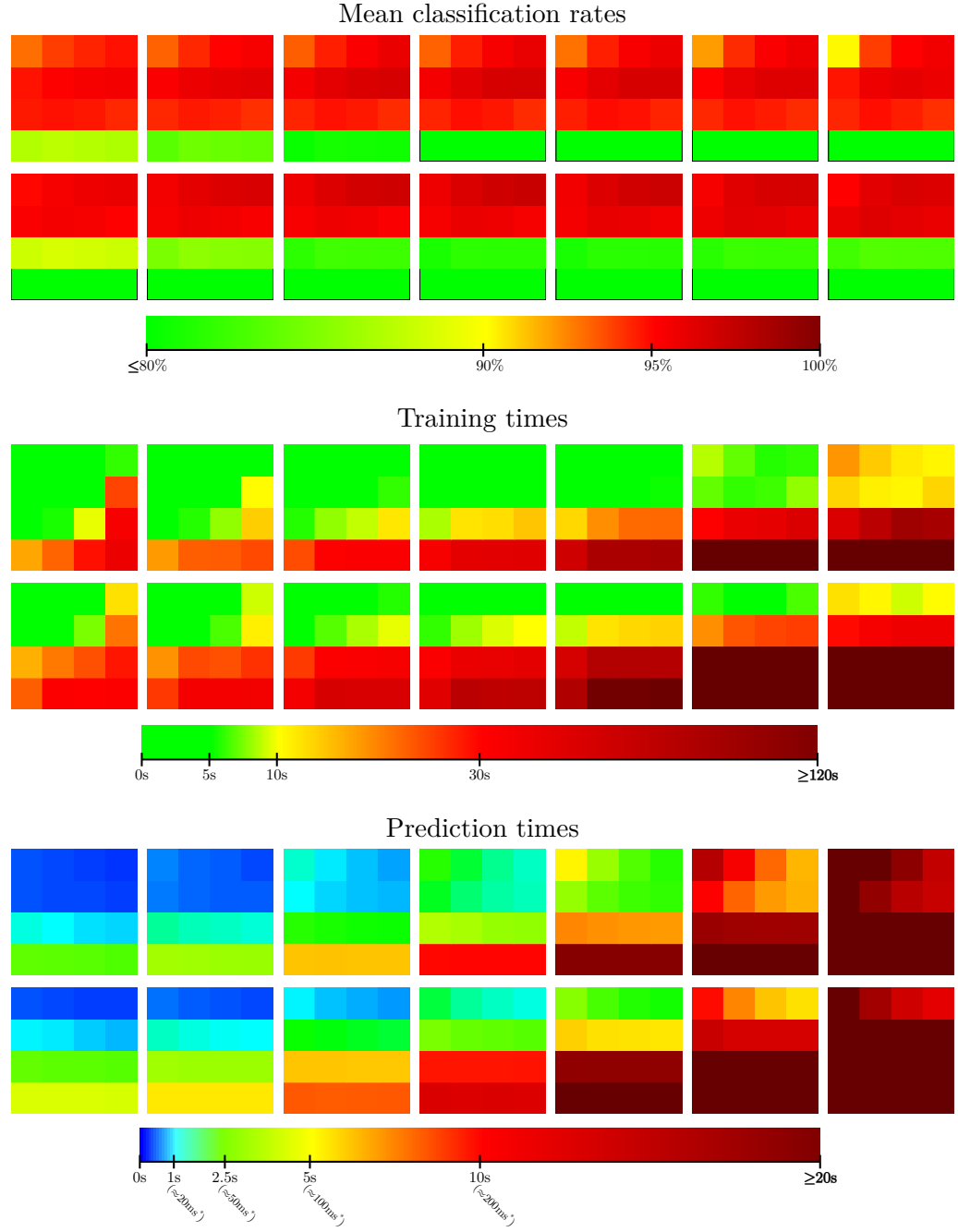


Figure 8.16: These histograms illustrate the differences when using sparse sampling in comparison to the intrinsic shape signature keypoint algorithm. The first row of each group is equivalent to the previously shown figures corresponding to ISS keypoints. The second row of each group shows the same results when using sparse sampled keypoints.

The next step is to compare the results according to the set of keypoints used. Figure 8.16 shows three different aspects of the support vector machines trained when using sparse sampled keypoints in comparison to the intrinsic shape signature keypoints. It shows that the mean classification rates, the training times as well as the prediction times have nearly the same values shifted by one γ -step.

In order to assess the visual results of the histograms, Table 8.14 contains the mean classification rates of ISS with $\gamma = 0.008$ (second row of each histogram) and of sparse sampling with $\gamma = 0.001$ (first row of each histogram).

Table 8.14: Classification results of ISS (i) and sparse sampling (s) for bag of features vocabularies with 10-100 entries. In case of ISS the values correspond to $\gamma = 0.0008$. In case of sparse sampling the values correspond to $\gamma = 0.0001$.

10/20	C	1	5	25	125	1	5	25	125
	i	94,66	95,09	95,34	95,47	95,22	95,75	96,04	96,15
	s	94,88	95,38	95,68	95,84	95,51	96,06	96,37	96,56
	$ i-s $	0,34	0,37	0,36	0,31	0,06	0,05	0,10	0,02
50/100	C	1	5	25	125	1	5	25	125
	i	95,45	96,12	96,48	96,58	95,46	96,18	96,57	96,65
	s	95,67	96,33	96,74	96,90	95,65	96,44	96,92	97,14
	$ i-s $	0,20	0,15	0,17	0,25	0,19	0,26	0,35	0,49

Comparing the results that could be achieved with ISS and sparse sampling shows that each best classification rate only depends on the values of C and γ while training the support vector machine. Furthermore, it is important to note that this characteristic holds not only for the mean classification rate, but also for the training and prediction time.

At this point another intermediate result can be summarized:

Considering the fact that the classification process is not influenced by the use of sparse sampling or the intrinsic shape signature keypoint algorithm, the intrinsic shape signature keypoint algorithm should be preferred, since the number of keypoints is much smaller in comparison to sparse sampling.

8.5.3 Local 3-D Feature Descriptors

Finally, a look at the results of different algorithms will be taken. In the previous sections it has turned out, that it is irrelevant which object classes have been used to determine the vocabulary, when the number of local 3-D feature descriptions becomes as large as in the context of this work. Thus, the vocabulary determined on the 10 object classes 'apple' to 'cellphone' will be used in the remainder of the work. Of course, one could also select the other two options. However, when taking the total calculation time into account, a smaller number of local 3-D feature descriptions leads to a faster computation of the bag of features clustering.

It has also been shown that the intrinsic shape signature keypoint algorithm is the better choice. While the classification rate is almost identical to the classification rate using sparse sampling, the number of keypoints and the number of feature descriptions, respectively, is much smaller. Therefore, the intrinsic shape signature keypoint algorithm and the vocabulary determined on the 10 object classes 'apple' to 'cellphone' are assumed as default in the following sections.

Figure 8.17 illustrates the results that can be achieved with different local 3-D feature description. It is particularly noticeable that the spin images have remarkable lower classification rates than all other algorithms. On the other hand it is not very surprising that independent of the algorithm that is used, the best results can always be found in the area of $C = 25$ and $\gamma = 0.0008$. This is of course, because the bag of features histograms are only statistics on the distribution of the feature vectors, so that it is not possible to get a relation to the feature descriptions used.

Finally, this raises the question: what about the prediction times for the different algorithms? An answer is provided in Figure 8.18. It appears, that there is a connection between the classification rates and the prediction times. Far more importantly, however, is the knowledge that each of the support vector machines is able to reach suitable classification rates within a reasonable computation time for the prediction.

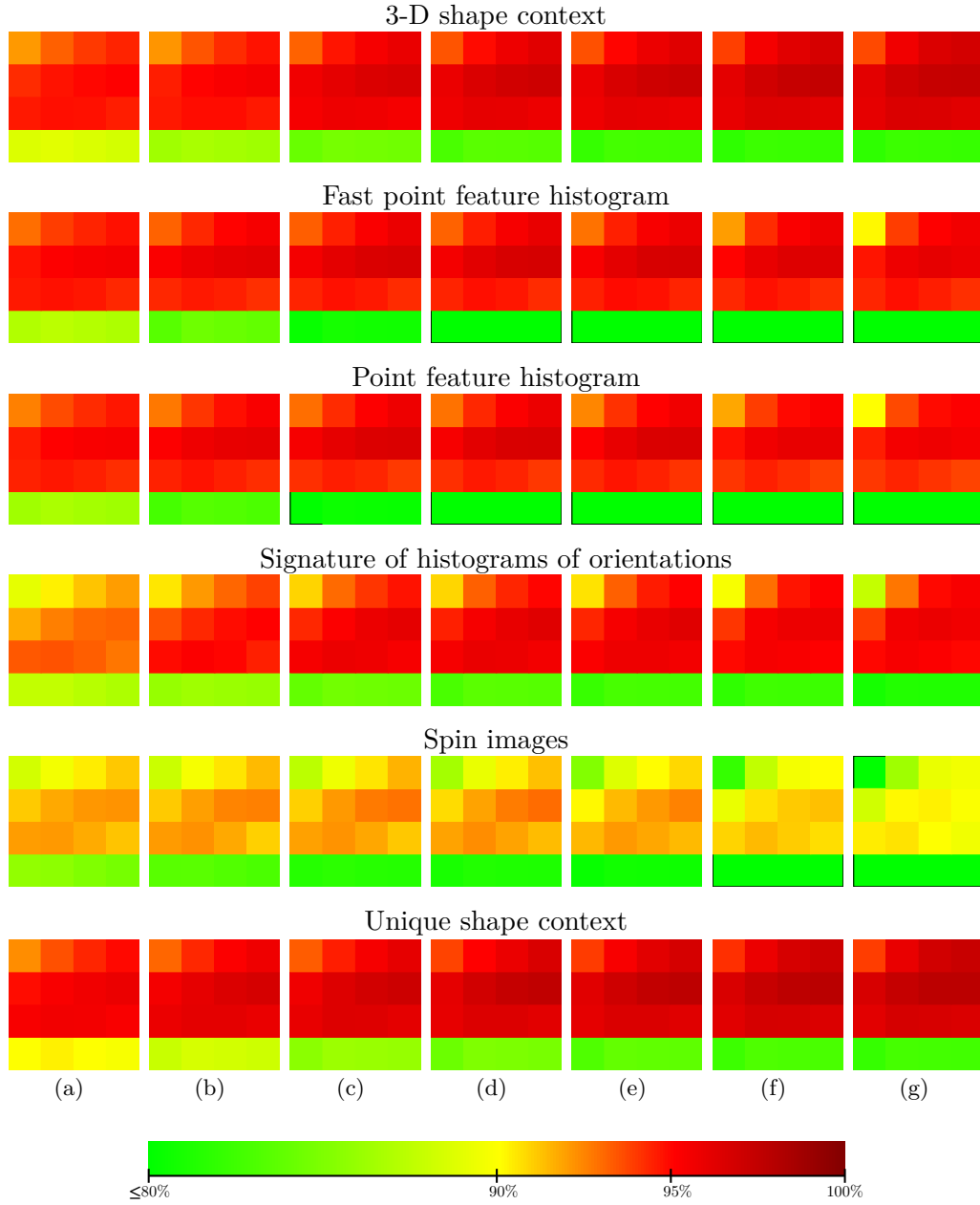


Figure 8.17: Mean classification rates for different local 3-D feature descriptors. As in all previous figures here it also applies that the columns (a)-(g) correspond to bag of features vocabularies with 10, 20, 50, 100, 200, 500, and 1000 entries.

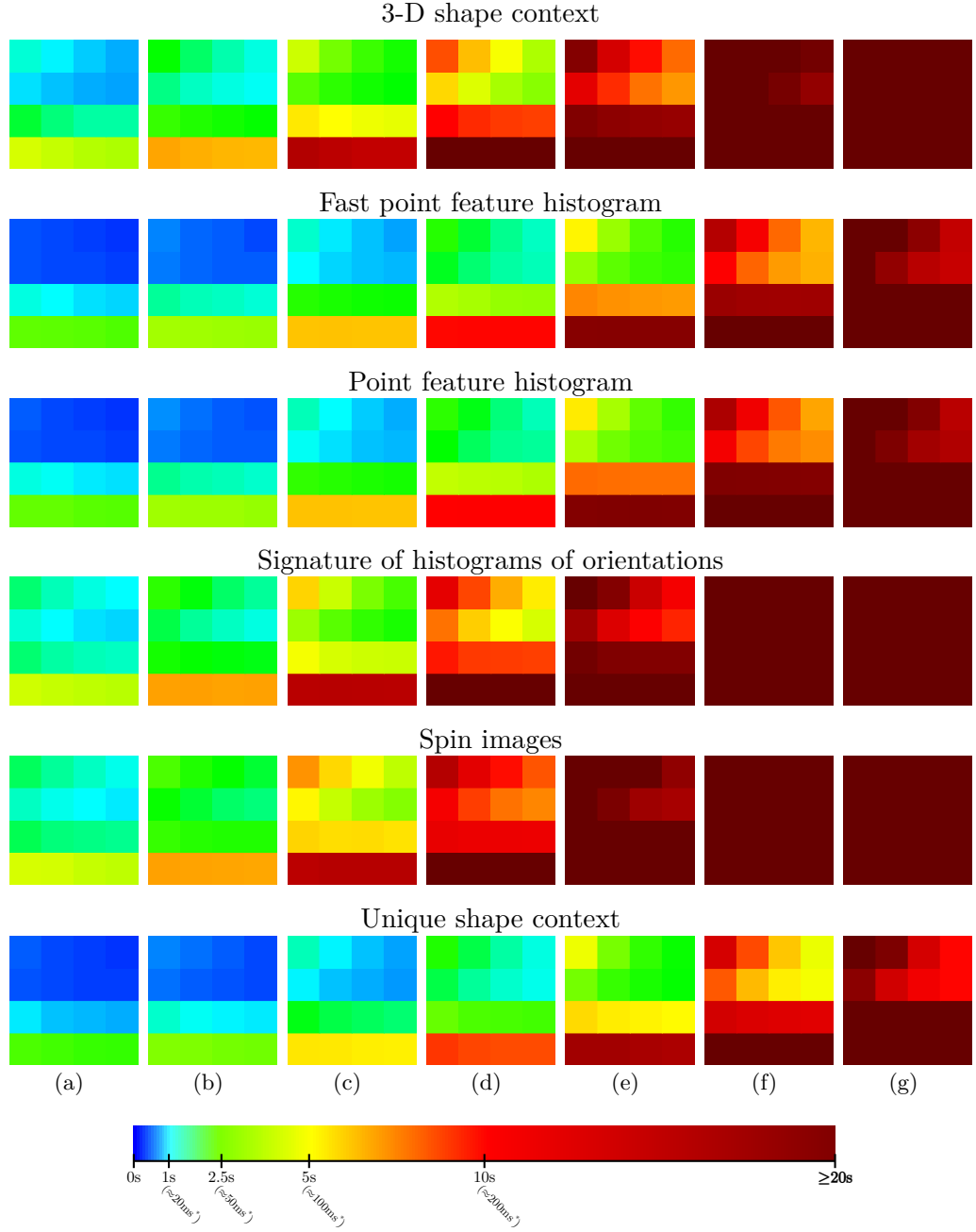


Figure 8.18: Prediction times for different local 3-D feature descriptors. As in all previous figures here it also applies that the columns (a)-(g) correspond to bag of features vocabularies with 10, 20, 50, 100, 200, 500, and 1000 entries. As noted in Figure 8.14 these values correspond to a prediction of ≈ 6000 input vectors. Hence the values in parentheses correspond to rough estimates for the required prediction times for a single prediction on all 51 object classes.

The parameters of the pipeline which are used in the remainder of this work will be selected as follows:

- support vector machines, which have the highest classification rates
- in conjunction with a prediction time limit for all object classes of $100ms$

This leads to the following set of parameters (Table 8.15):

Table 8.15: Final set of pipeline parameters / support vector machines

Features	Keypoints	BoF clustering	BoF size	C	γ	Classifica- tion rate	Predic- tion time
3DSC	ISS	10 cl.	100	125	0.008	96.97%	$3.62s (\approx 72ms^*)$
FPFH	ISS	10 cl.	100	125	0.008	96.65%	$1.30s (\approx 26ms^*)$
PFH	ISS	10 cl.	100	125	0.008	96.56%	$1.40s (\approx 28ms^*)$
SHOT	ISS	10 cl.	100	125	0.008	96.27%	$4.52s (\approx 90ms^*)$
SI	ISS	10 cl.	50	125	0.008	92.80%	$3.57s (\approx 71ms^*)$
USC	ISS	10 cl.	200	125	0.008	97.62%	$2.08s (\approx 42ms^*)$

8.5.4 Precision and Recall

In order to evaluate a classifier besides the pure classification rate, which is usually referred to as recall, the accuracy also known as precision is also important. The values of the following precision-recall-diagrams will be determined by the definitions shown in Table 8.16.

Table 8.16: Terminology of the terms true positives, true negatives, false positives, and false negatives in context of classification tasks.

true positive (t_+)	an object corresponding to object class C is correctly assigned to C
true negative (t_-)	an object corresponding to object class C is correctly rejected from another objects class C'
false positive (f_p)	an object corresponding to object class C is incorrectly assigned to another objects class C'
false negative (f_n)	an object corresponding to object class C is incorrectly rejected from C

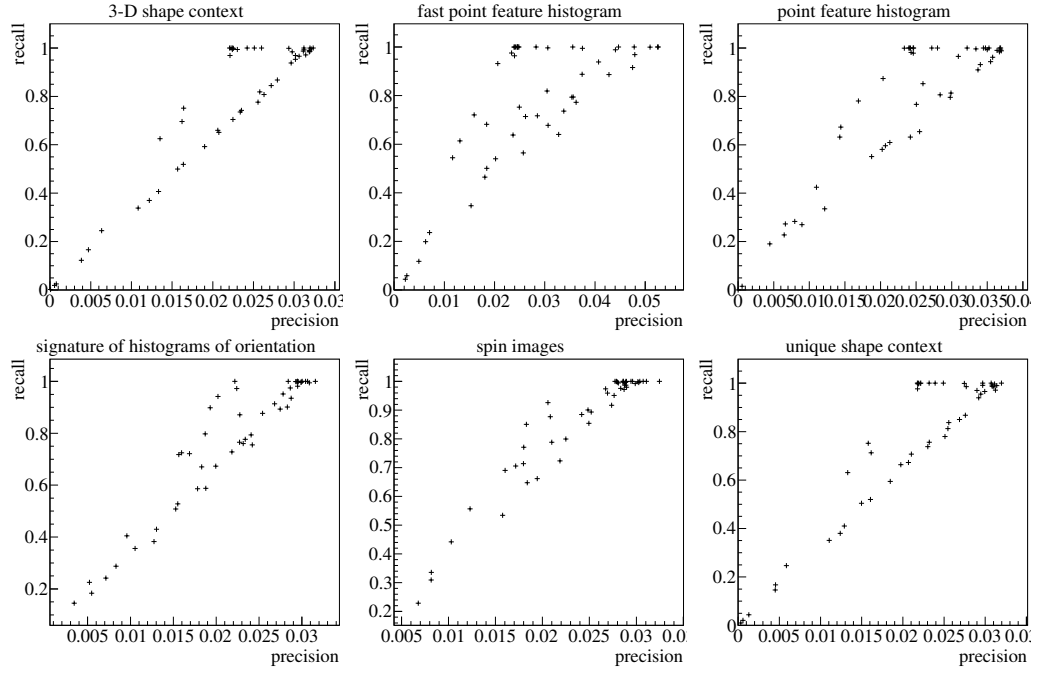


Figure 8.19: Precision-recall graphs for all classifiers. Each data point (+) corresponds to a single object class.

Precision and recall are then defined as [62]

$$\begin{aligned}
 \text{precision} &= \frac{t_+}{t_+ + f_+}, \\
 \text{recall} &= \frac{t_+}{t_+ + f_-}.
 \end{aligned}
 \tag{8.4}$$

In the following diagrams precision and recall are presented for each individual local 3-D feature description algorithm. Each data point corresponds to a single object class.

The graphs shown in Figure 8.19 illustrate that in particular the precision values of many object classes are very low. However, that was to be expected, because the shapes of many objects are too similar to each other.

8.5.5 Increasing the Limit of the Prediction Values

The question is whether it is possible to increase the precision at least slightly. To achieve this it was tested how precision and recall are changing if the limit

for the assignment to a particular class is increased successively. The results are in Table 8.17.

Table 8.17: This table shows the precision (pre.) and recall (rec.) values of all local 3-D feature description algorithms for different classification limits (lim.). The latter corresponds to the distance to the hyperplane, which separates the two classes of a support vector machine to an input vector, i.e., a bag of features histogram. Thus, only those objects which result in a distance to the hyperplane greater than the limit are assigned to the corresponding object class.

	3dsc		fpfh		pfh		shot		si		usc	
lim.	pre.	rec.	pre.	rec.	pre.	rec.	pre.	rec.	pre.	rec.	pre.	rec.
2.00	3.18	68.17	4.55	69.79	3.81	71.82	2.89	73.25	3.33	83.78	3.18	72.75
3.00	3.38	65.05	5.15	64.78	4.30	68.29	3.31	69.98	3.71	82.64	3.38	68.17
4.00	4.01	61.79	5.62	60.84	4.56	65.61	3.59	67.05	3.91	81.32	4.01	65.05
5.00	4.07	58.96	5.75	57.39	5.24	62.96	4.16	65.27	4.12	79.76	4.01	61.79
6.00	4.17	55.43	6.63	54.52	6.10	60.24	4.20	63.78	4.22	77.98	4.41	58.96

This shows that the growth of the mean precision unfortunately has a limited extent. This is also not surprising, because this way it is not possible to differentiate similar objects. However, there is a certain number of object classes in the dataset that should allow a better differentiation. To this end, the number of object classes with a precision $\geq 5\%$ and a recall $\geq 75\%$ will be counted while increasing the classification limit. The corresponding results are shown in Table 8.18. The best limit-values are highlighted gray.

Table 8.18: Number of object classes with a precision $\geq 5\%$ and a recall $\geq 75\%$ for different classification limits.

	3dsc	fpfh	pfh	shot	si	usc
1.0	0	5	0	0	0	0
2.0	8	19	10	0	2	8
3.0	8	18	9	5	5	8
4.0	8	18	7	5	6	8
5.0	8	17	6	6	7	8
6.0	10	16	7	5	5	10
7.0	13	15	6	4	6	13
8.0	14	14	7	4	6	14
9.0	13	14	9	4	7	13

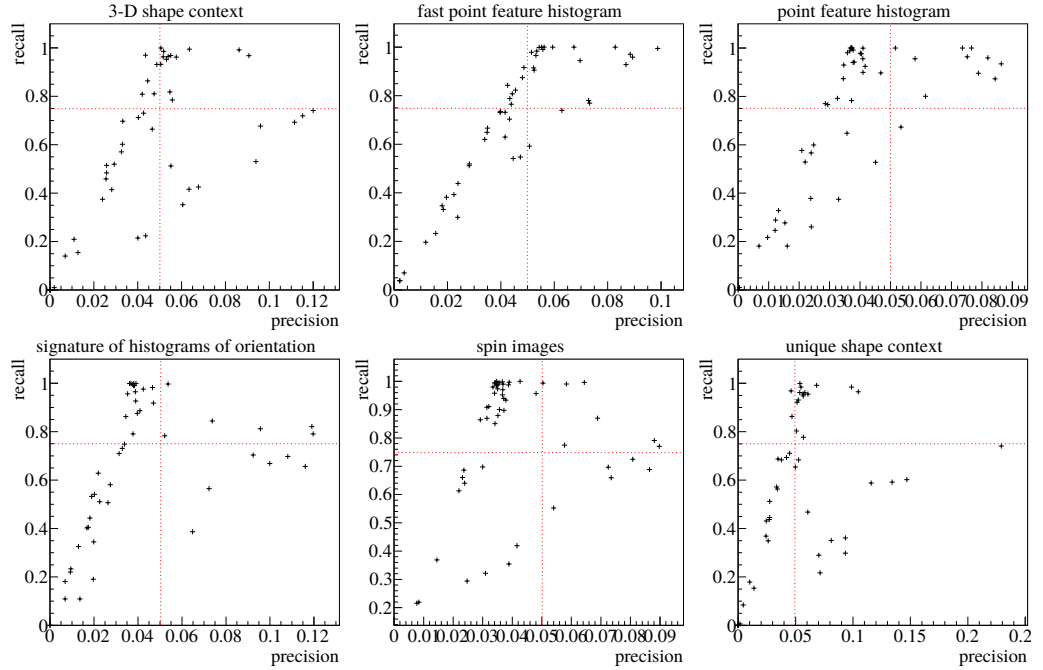


Figure 8.20: Precision-recall graphs for all classifiers with different limits. The red dotted lines depict the counting limits of 5% precision and 75% recall. The corresponding limits are taken from Table 8.18: 3DSC: 8.0, FPFH: 2.0, PFH: 2.0, SHOT: 5.0, SI: 5.0, and USC: 8.0.

The corresponding precision-recall graphs for the highlighted limit values are shown in Figure 8.20.

During the learning phase of the reinforcement learning algorithm both variants of support vector machines, i. e., the support vector machines with a regular distance of 0.0 to the separating hyperplane and the support vector machines with a distance corresponding to the highlighted values in Table 8.18 will be made available.

8.5.6 Reducing the Number of Classes

Due to the very small precision values it can be assumed that an unambiguous assignment of an object to a single object class is almost impossible. To determine the classification rate of a single local 3-D feature description algorithm in relation to the classification rate of a sequence of algorithms, a subset of 10 of the 51 classes of objects is determined. This set is determined from the object classes with the best precision values. Table 8.19 shows the set.

Table 8.19: Subset of 10 object classes.

class no.	object class
8	cap
11	coffee_mug
15	food_bag
22	greens
23	hand_towel
25	keyboard
26	kleenex
32	notebook
37	pitcher
43	shampoo

8.5.7 Interim Conclusion on Training and Classification Results

Table 8.20 shows the classification results of single local 3-D feature description algorithms when used with the basic classification pipeline. The six types of terminal states in the order of their tests are:

1. If the set of class candidates does not contain the corresponding object class, the terminal state is counted as *fail state*.
2. If the computation time limit was exceeded two sub-cases can be distinguished:
If the class candidate with the maximum of summed prediction values over all algorithms that have been applied
 - (a) corresponds to the correct object class, the state is counted as *overtime/match*.
 - (b) does not correspond to the correct object class, the state is counted as *overtime/miss*.
3. If the class candidates contains only a single object class, the state is counted as an *exact match*.
4. Finally, if no algorithm is left, another two sub-cases can be distinguished:
If the class candidate with the maximum of summed prediction values over all algorithms that have been applied

- (a) corresponds to the correct object class, the state is counted as *no actions/match*.
- (b) does not correspond not to the correct object class, the state is counted as *no actions/miss*.

As already introduced in Section 7.3.3, a time limit of 10 seconds is assumed for the subsequent results.

Table 8.20: Results of single algorithms when applied to objects of the complete data set. None of the algorithms is able to classify a single object without uncertainty. The point feature histogram is so slow, that about 17 to 18% of all objects result in one of the two overtime states, i. e., require more than 10 seconds for computation.

algorithm	3DSC	3DSC	FPFH	FPFH	PFH	PFH
classification limit	0.0	8.0	0.0	2.0	0.0	2.0
fail state	24.3%	51.1%	21.8%	28.5%	23.8%	28.1%
overtime/match	0.0%	0.0%	0.0%	0.0%	2.8%	2.8%
overtime/miss	0.0%	0.0%	0.0%	0.0%	15.2%	14.0%
exact match	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
no actions/match	6.1%	5.7%	9.4%	9.7%	3.2%	2.9%
no actions/miss	69.6%	43.2%	68.8%	61.8%	55.0%	52.2%
sum of positive results	6.1%	5.7%	9.4%	9.7%	6.0%	5.8%

algorithm	SHOT	SHOT	SI	SI	USC	USC
classification limit	0.0	5.0	0.0	5.0	0.0	8.0
fail state	27.3%	37.7%	15.5%	20.3%	22.2%	43.0%
overtime/match	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
overtime/miss	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
exact match	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
no actions/match	3.6%	3.8%	7.4%	7.3%	8.5%	8.7%
no actions/miss	69.1%	58.5%	77.1%	72.4%	68.2%	47.3%
sum of positive results	3.6%	3.8%	7.4%	7.3%	8.5%	8.7%

It shows, that it is not possible to assign an object to a single object class. Apart from this, the ratio of sets of class candidates, which contains the correct object class (overtime/match + hit + out-of-algorithm/match), is between $\approx 50\%$ and $\approx 80\%$. Interestingly, the point feature histogram (PFH) is so slow, that

about 17 to 18% of all objects result in one of the two overtime states, i. e., require more than 10 seconds for computation.

Table 8.21: Results of single algorithms when applied to objects of the reduced data set of 10 object classes.

algorithm	3DSC	3DSC	FPFH	FPFH	PFH	PFH
classification limit	0.0	8.0	0.0	2.0	0.0	2.0
fail state	7.9%	63.0%	13.2%	17.9%	14.0%	18.2%
overtime/match	0.0%	0.0%	0.0%	0.0%	52.5%	51.2%
overtime/miss	0.0%	0.0%	0.0%	0.0%	14.3%	12.4%
exact match	0.0%	17.1%	0.0%	7.9%	0.0%	0.7%
no actions/match	56.6%	12.2%	65.0%	57.7%	10.4%	9.5%
no actions/miss	35.5%	7.7%	21.8%	16.5%	8.7%	8.1%
sum of positive results	56.6%	33.3%	65.0%	65.5%	62.9%	61.4%

algorithm	SHOT	SHOT	SI	SI	USC	USC
classification limit	0.0	5.0	0.0	5.0	0.0	8.0
fail state	19.0%	43.0%	15.3%	20.4%	3.6%	60.0%
overtime/match	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
overtime/miss	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
exact match	0.0%	5.1%	0.0%	0.8%	0.0%	18.1%
no actions/match	22.8%	18.6%	23.8%	23.2%	59.7%	15.2%
no actions/miss	58.1%	33.3%	60.9%	55.6%	36.6%	6.7%
sum of positive results	22.8%	23.7%	23.8%	23.9%	59.7%	33.3%

In comparison, the results using the reduced number of object classes are shown in Table 8.21. Although the results still do not show a considerable number of states that lead to an exact match of the object class, at least low rates in which the object classes can be identified with a single algorithm can be achieved. On the other hand the number of hits in the sets of class candidates increased significantly. But given the small number of object classes this is not surprising.

8.6 Conclusion

That the classification rates of individual objects are so low in comparison to the recognition rates reported in the corresponding evaluations shows how important it is to find a way to increase the classification rates.

A possible reason for this problem might be the major principle underlying the classification task. While the computational effort for a one-to-one comparison of the local 3-D feature descriptions in the context of object recognition is acceptable, this approach is not acceptable for large scale classification tasks. For this reason, the bag of features approach is used to aggregate the local 3-D feature descriptions to a generic and even smaller description. This involves that essential information of local feature descriptions falls a victim to statistics. However, the latter has only a chance to be as informative as possible, if the number of counted elements is relatively large.

At this point the question from the last paragraph of Section 8.4.1, i. e., the construction of the bag of features vocabulary should be addressed once again. There it was asked if there might be a correlation between the long computation time of k-means in the determination of the vocabulary for 3DSC and the informativeness of the local 3-D feature vectors. It was assumed that this might be reflected in the classification results. This question can be answered unequivocally with “no”, since the classification results between 3DSC and other algorithms such as PFH and USC hardly differ.

Accordingly, the informativeness of the bag of feature histograms has to be gradually increased by a skillfully combination of different algorithms, which will be implemented in the following by the fusion of the basic classification pipeline with a reinforcement learning framework.

9

Adaptive 3-D Object Classification with Reinforcement Learning

In this chapter, the primary results of this work are presented. The structure of the chapter is geared to the outline proposed in Section 7.2.2 to Section 7.2.4.

The first section will demonstrate that learning of a sequence of local 3-D feature description algorithms leads to significantly better classification results.

The second section shows, how global properties of a 3-D point cloud can help the reinforcement learning agent to select a proper initial action to reduced the number of fail states after the first application of an algorithm.

The third section shows how the ϵ -greedy episodes work and how the classification can benefit from a new feature descriptor added afterwards, when this descriptor provides better features and thus leads to better classification results.

9.1 Fusion with Reinforcement Learning

The reinforcement learning framework initially consists of the elements introduced in Section 7.2.2. To keep it in mind, Figure 9.1 illustrates the corresponding model again.

As already introduced in Section 7.3.3, a time limit of 10 seconds is assumed for the subsequent results. The following graphs show different aspects from the course of learning of the reinforcement learning framework. In each episode the learner passes through the following steps:

1. The set of object class candidates $\mathcal{C}$ is initialized with all available object classes.

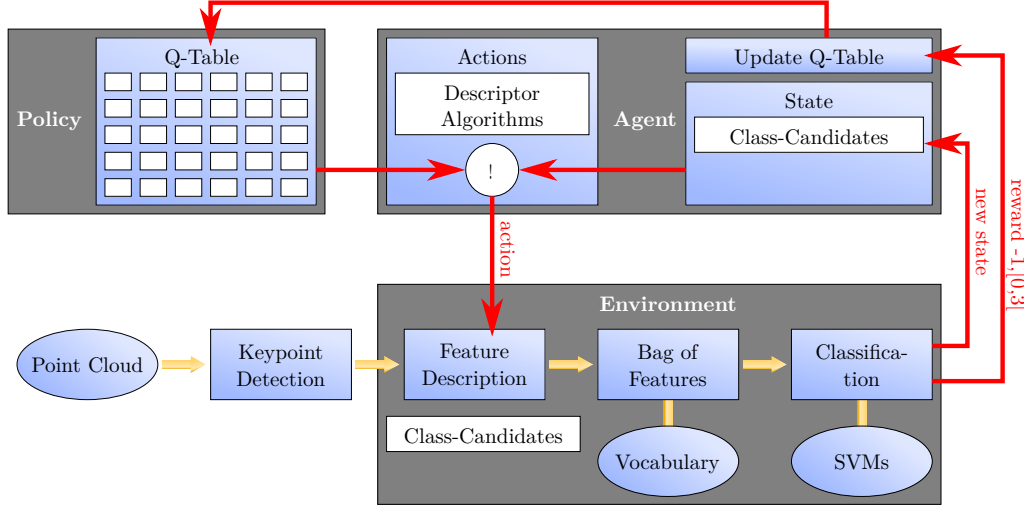


Figure 9.1: Model I: the fusion of the basic classification pipeline with reinforcement learning.

2. The set of algorithms $\mathcal{A}$ is initialized with all available local 3-D feature description algorithms.
3. An object o is selected randomly from all available objects $\mathcal{O}$, $|\mathcal{O}| = 207481$.
4. A local 3-D feature description algorithm $a \in \mathcal{A}$ is selected randomly. Later, a parameter ϵ controls the probability of a random choice of an algorithm.
5. Algorithm a is applied to the 3-D point cloud of the selected object o .
6. For each of the remaining class candidates $c \in \mathcal{C}$ a classification is done with the corresponding support vector machine and the final 3-D object description, i.e., the bag of features histogram.
7. Based on the classification results and depending on the limit of the prediction values for the currently selected algorithm the class candidate c is kept or removed from $\mathcal{C}$.
8. Finally, it is checked whether the current state is a terminal state:
 - (a) The set of class candidates does not contain the correct class:
 $\rightarrow$ *fail state*, reward $r = -1.0$
 - (b) The computation time limit was exceeded, max prediction sum
 - i. belongs to the correct object class:
 $\rightarrow$ *overtime/match*, reward $r \in [1.0, 2.0[$

- ii. does not belong to the correct object class:
 $\rightarrow$ *overtime/miss*, reward $r \in [0.0, 1.0[$
- (c) One class left:
 $\rightarrow$ *exact match*, reward $r \in [2.0, 3.0[$
- (d) No algorithms left unused, max prediction sum
 - i. belongs to the correct object class:
 $\rightarrow$ *no actions/match*, reward $r \in [1.0, 2.0[$
 - ii. does not belong to the correct object class:
 $\rightarrow$ *no actions/miss*, reward $r \in [0.0, 1.0[$

If the state is none of the terminal states above, the reinforcement learning agent continues with step 3.

9. If the learning rate $\alpha > 0$ the Q -value is updated.

The cases (b).i, (c), and (d).i are the three cases that would lead to a correct classification.

Subsequently, four different graphs visualize the course of the learning phase:

- The first graph illustrates the number of states stored in the Q -table. This value is mainly of interest because it reflects the real number of different states inspected by the reinforcement learning framework, since, as already mentioned, the theoretical number of states is magnitudes larger ($2^{51} \cdot 2^n$, where n is the number of actions).
- The second graph shows the average Q -value over all state-action-pairs. This value indicates if the Q -table is becoming stable, and thereby roughly the number of episodes after which the learning rate α and the ϵ -value can be lowered.
- The third graph shows the classification rates throughout the entire process. Strictly speaking, the graph shows the percentage of the 6 possible terminal states (see above) over time.
- The last graph shows the percentage of the number of actions used in each episode until the terminal was reached.

In all graphs the dotted blue line illustrates the learning rate α , while the dashed blue line corresponds to the ϵ -value.

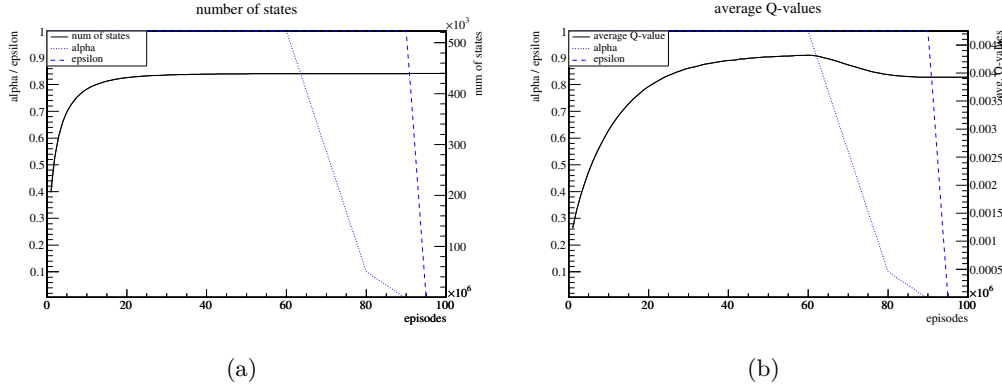


Figure 9.2: These two figures show the number of states (a) and the average Q-value (b) – both are depicted by the continuous black line and their axes on the right side. The blue lines show the values of the learning rate α (dotted) and the ϵ -value (dashed) with the axes on the left side.

In the following subsections the results of different configurations will be compared:

1. The use of all object classes and the use of 3 different algorithms in configurations with two different limits. To be precise:
PFH/0.0, PFH/2.0, SHOT/0.0, SHOT/5.0, USC/0.0, USC/8.0.
2. The use of 10 object classes and the use of 3 different algorithms in configurations with two different limits. To be precise:
PFH/0.0, PFH/2.0, SHOT/0.0, SHOT/5.0, USC/0.0, USC/8.0.
3. The use of all object classes and the use of 6 different algorithms with a prediction value limit of 0.0:
3DSC, FPFH, PFH, SHOT, SI, USC.
4. The use of 10 object classes and the use of 6 different algorithms with a prediction value limit of 0.0:
3DSC, FPFH, PFH, SHOT, SI, USC.

9.1.1 Results for all Object Classes – Different Limits

The following results illustrate the classification results that can be achieved without the use of global information in the initial state.

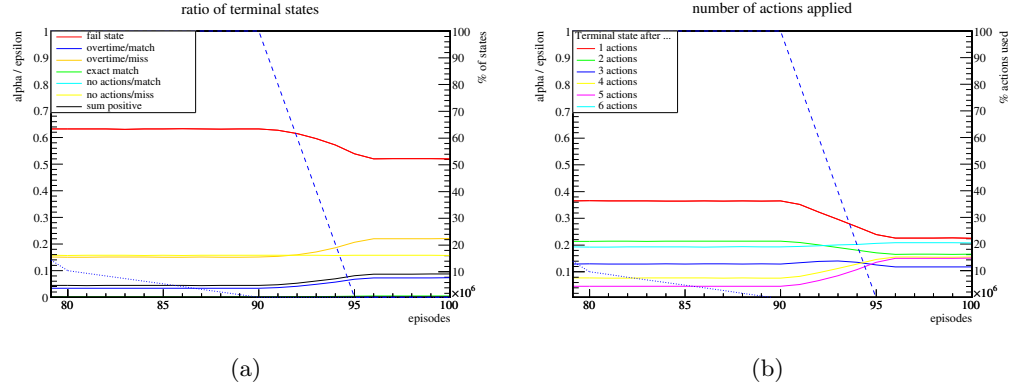


Figure 9.3: Influence of the learned Q -values on the classification results. Sub-figure (a) shows the proportion of the 6 different terminal states (the colored lines) and the sum of positive states (black line). Sub-figure (b) shows the number of actions that have been applied until a terminal state was reached.

Figure 9.2 illustrates the number of states stored in the Q -table and the average Q -value for a single state during the initial learning phase of the reinforcement learning framework. The set of algorithms available consists of PFH, SHOT and USC with 2 different limits for the prediction values each.

Referring to Figure 9.2 (a) it can clearly be seen that the real number of states (in this case ≈ 450000) is several orders of magnitude smaller than the theoretical number of states, i. e., all possible subsets of the object classes times all possible subsets of already applied algorithms: $2^{51} \cdot 2^6$. The reason for this is that there are de facto no larger subsets with more than 10 class candidates. Actually, this leads to a fast saturation of the number of states after an acceptable number of about 50 million episodes. These 50 million episodes take approximately 30 seconds to compute.

The second figure (Figure 9.2 (b)) shows the change of the mean Q -values over all state-action pairs. The stagnation of these values means, that the Q -values alter slightly. Therefore, this is the point (after 60 million episodes) where the learning rate α is continuously reduced (the blue dotted line). The slight reduction of the average Q -value shows that the Q -values were a little bit over fitted. But they keep stable with a decreasing learning rate. Finally, when the learning rate is zero (after 90 million episodes) the ϵ -value, i. e., the ratio of randomly selected actions is reduced from 1.0 (random-policy) to 0.0 (max Q -policy).

The classification results that can be achieved with this reinforcement learning model are illustrated in Figure 9.3. Only the episodes from 80 to 100 million are shown, since the classification results of a purely random selection (< 90 million episodes) are all identical. Figure 9.3 (a) shows the six different terminal states. One can see, that the classification results increase with a decreasing ϵ -value. To get a better picture, all relevant values are shown in Table 9.1.

Table 9.1: This table shows the change of the classification results in relation to Figure 9.3. The first column corresponds to episode 90 million, i. e., the last episode with 100% randomly selected actions ($\epsilon = 1.0$). The second column corresponds to episode 100 million, where a maxQ-policy is used ($\epsilon = 0.0$).

	random	maxQ	change
episodes	90 M	100 M	(rel.)
fail state	63.4%	52.2%	-17.6%
overtime/match	3.8%	7.8%	102.4%
overtime/miss	15.6%	22.4%	44.2%
exact match	0.6%	1.0%	56.4%
no actions/match	0.4%	0.4%	-0.6%
no actions/miss	16.2%	16.1%	-0.1%
sum of positive results	4.8%	9.2%	87.3%

The combination of the three different algorithms leads to a higher classification rate in each case (PFH = 6.0%/5.8%, SHOT = 3.6%/3.8%, and USC = 8.5%/8.7%). Moreover, for the first time within this work it is possible to achieve exact assignments. However, the improvement over the best single algorithm (USC) can be referred to as marginal.

Looking at Figure 9.3 (b) shows that the distribution of the number of terminal states is relatively uniform, and that the two cases, where the reinforcement learning frameworks stops after the first action (red) and all actions (cyan) have the largest share.

In the latter case several situations may have occurred. The most common case is the situation that the slowest algorithm (here PFH) was carried out at the end. In most cases this led to an overtime terminal state. This shows that the choice of the time limit is relatively well suited. Therefore, it makes only sense to reduce the time limit if necessary.

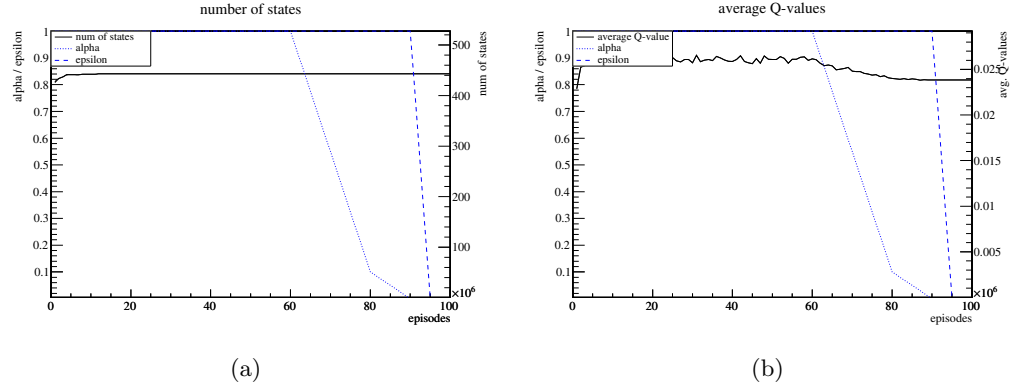


Figure 9.4: These two figures show the number of states (a) and the average Q-value for a single state (b) for a smaller number of 10 object classes.

More interestingly, however, is the interpretation of the first case. This case can be achieved only if the result is an exact match of the object class or a fail state. In addition, the amount of these cases is drastically decreasing while reducing ϵ , which means randomly chosen fail states dominate.

9.1.2 Results for 10 Object Classes – Different Limits

In this section the results which can be achieved without the use of global information in the initial state are considered in the same manner. The only difference is, that not all but only 10 object classes are used (see Section 8.5.6).

Figure 9.4 shows the number of states and the average Q-value per state. During this experiment, deliberately the same number of episodes was used, to make a direct comparison of the learning curve. It turns out that the small number of 443 states means that the learning curves converge much faster, i. e., within 5 million episodes.

Analogously, the classification results that can be achieved with this reinforcement learning model are illustrated in Figure 9.5. Here, the classification rates are expected to be significantly higher. However, the $\approx 65\%$ positive terminal states are similar to the best individual results of the algorithms used (PFH achieved classification rates of 62.9% and 61.4%).

To get here a better overview, all relevant values are shown in Table 9.2.

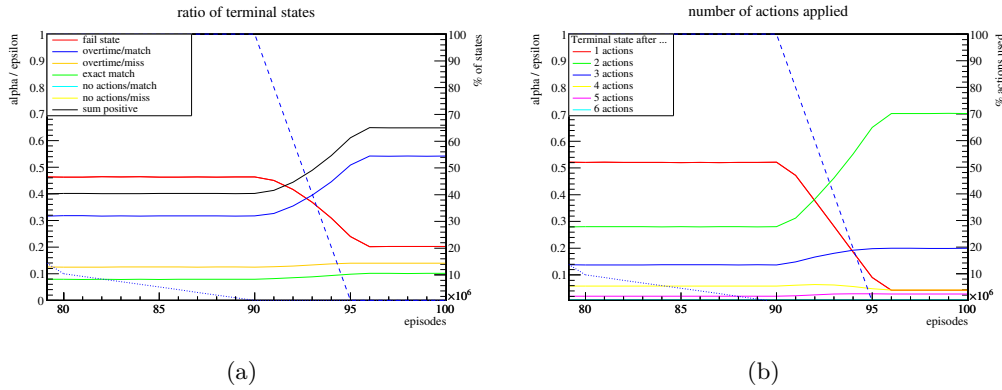


Figure 9.5: These two figures show the classification results when using only 10 object classes.

Table 9.2: This table shows the change of the classification results in relation to Figure 9.5. The first column corresponds to episode 90 million, i. e., the last episode with 100% randomly selected actions ($\epsilon = 1.0$). The second column corresponds to episode 100 million, where a maxQ-policy is used ($\epsilon = 0.0$).

	random	maxQ	change
episodes	90 M	100 M	(rel.)
fail state	46.6%	20.6%	−55.8%
overtime/match	32.1%	54.4%	69.6%
overtime/miss	13.0%	14.4%	11.0%
exact match	8.3%	10.6%	26.6%
no actions/match	0.0%	0.0%	0.0%
no actions/miss	0.0%	0.0%	0.0%
sum of positive results	40.4%	65.0%	60.8%

But there is a major difference in comparison to the results from the previous section: the distribution of the number of actions. Almost all episodes end after using 2 or 3 actions. Together, the two cases arise approximately 90%. In fact, the action which is always selected in the first state is USC performed with a limit of 0.0. According to Table 8.21 this is precisely the algorithm that achieves by far the smallest percentage of fail states: 3.6%. This corresponds exactly to the value of the graph in Figure 9.5 for a single action. And it shows that the reinforcement learning agent was able to make an optimal decision.

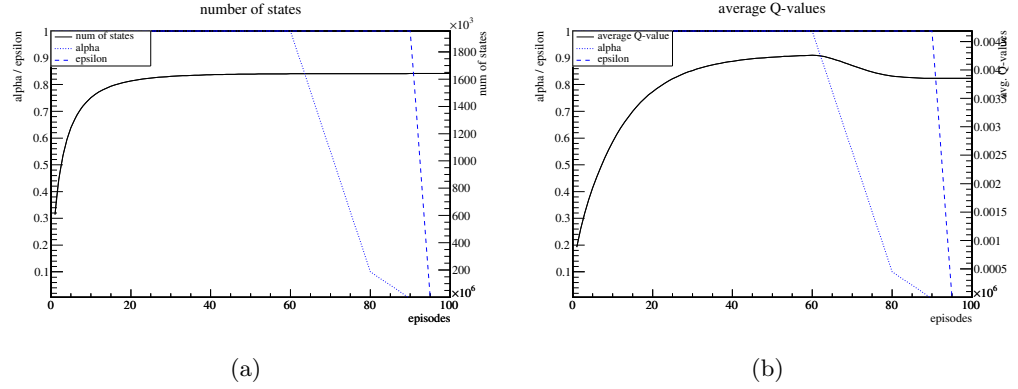


Figure 9.6: These graphs show the number of states (a) and the average Q -value during the reinforcement learning. All 6 algorithms are available as possible actions that can be selected by the reinforcement learning agent. During classification only the standard limit of 0.0 is used.

9.1.3 Results for all Object Classes – Unique Limits

The classification results that can be achieved without the use of global information in the initial state and with a “normal” usage of support vector machines, i. e., with the separation of the classes at a prediction value of 0.0 are shown in the following sections. Instead of three algorithms with two configurations, all six algorithms will be used.

The comparison begins in an analogous manner as in the previous sections with the figures of the number of states (Figure 9.6 (a)) and the average Q -value (Figure 9.6 (b)).

At first it can be noticed, that the number of 1640769 actual states is almost four times as high as shown in Figure 9.2 (a). This can be explained as follows: While the set of the class candidates for an algorithm with a higher prediction limit must always be a subset of the class candidates of the same algorithm with a limit of zero, there isn’t a similar relationship between the 6 different algorithms in the current case. Thus, the variety of permutations within the class candidates is correspondingly higher. Apart from that, the graphs differ only marginally from those in Figure 9.2.

In principle, the same applies also for the classification rates, which are shown in Figure 9.7. However, it turns out, that the classification rates that can be achieved with the approach proposed in this section can be clearly distinguished

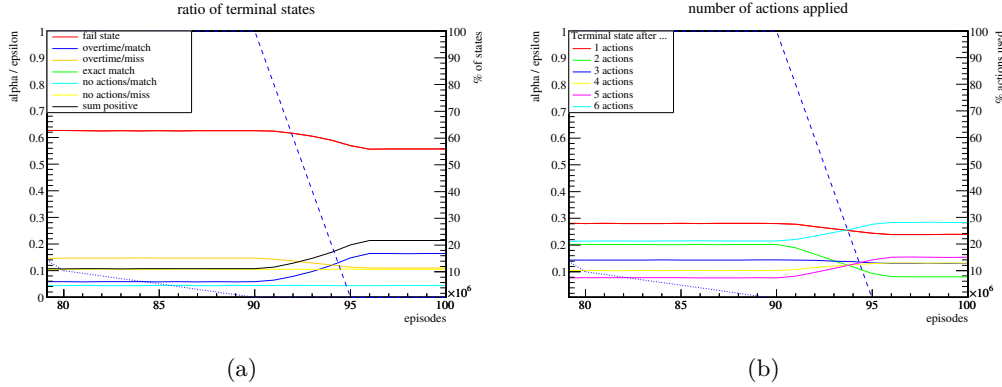


Figure 9.7: Influence of the learned Q-values on the classification results. Sub-figure (a) shows the proportion of the 6 different terminal states (the colored lines) and the sum of positive states (black line). Sub-figure (b) shows the number of actions that have been applied until a terminal state was reached. The actions that have been used by the reinforcement learning agent correspond to the 6 available algorithms while the classification was done at the standard limit of 0.0 only.

from the classification rates of individual algorithms or the approach introduced in Section 9.1.1. While the best individual algorithm (FPFH with a limit of 0.0) achieves a classification rate with 9.4% positive terminal states and the result from Section 9.1.1 reaches only 9.2%, the rate of nearly 22% positive terminal states that can be achieved here is almost 2.5 times higher. The only negative aspect might be the number of exact matches, since this value is still zero.

The exact values are as usual summarized in Table 9.3.

9.1.4 Results for 10 Object Classes – Unique Limits

Consequently, the consideration of the results for 10 object classes follows the same scheme. Accordingly, the section starts with the graphs on the number of states (Figure 9.8 (a)), and the average Q-value (Figure 9.8 (b)).

Figure 9.8 (a) clearly shows how quickly the number of states converges to a value of 571. In addition, the Q -values converge fairly quick. However, since the number of states is relatively low, the values fluctuate slightly with the results of each episode. With a learning rate of 1.0 that was to be expected and decreases parallel to the reduction of the α -value.

Table 9.3: This table shows the change of the classification results in relation to Figure 9.7. The first column corresponds to episode 90 million, i.e., the last episode with 100% randomly selected actions ($\epsilon = 1.0$). The second column corresponds to episode 100 million, where a maxQ-policy is used ($\epsilon = 0.0$). The actions that have been used by the reinforcement learning agent correspond to the 6 available algorithms while the classification was done at the standard limit of 0.0 only.

	random	maxQ	change
episodes	90 M	100 M	(rel.)
fail state	62.7%	55.9%	−10.9%
overtime/match	6.3%	16.8%	168.3%
overtime/miss	15.1%	11.4%	−24.6%
exact match	0.0%	0.0%	0.0%
no actions/match	4.9%	4.9%	0.4%
no actions/miss	11.0%	10.9%	−0.2%
sum of positive results	11.2%	21.7%	94.9%

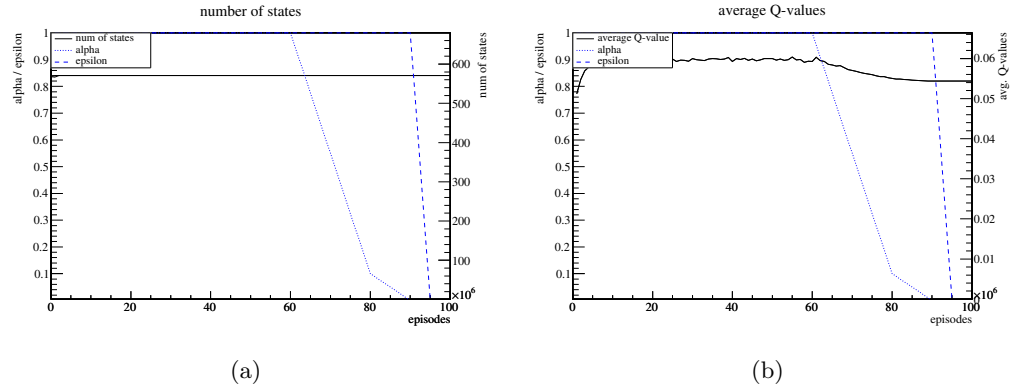


Figure 9.8: These two figures show the number of states (a) and the average Q-value for a single state (b) for a smaller number of 10 object classes. The actions that have been used by the reinforcement learning agent correspond to the 6 available algorithms while the classification was done only at the standard limit of 0.0.

Quite positive results can be found in the classification results shown in Figure 9.9. Using the current method it is possible to raise the rate of positive terminal states to over 73%. In addition, the proportion of exact assignments is over 15% – and that, although it was not possible to get a single exact match with each individual algorithm on the reduced set of 10 object classes.

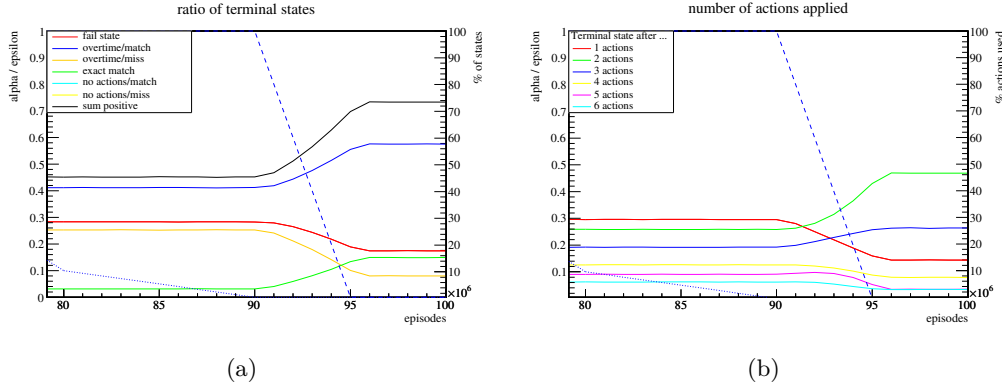


Figure 9.9: These two figures show the classification results when using only 10 object classes. The actions that have been used by the reinforcement learning agent correspond to the 6 available algorithms while the classification was done at the standard limit of 0.0 only.

A close look at the data also shows that the 17.9% remaining fail states are broadly caused by false negatives after the first action. It remains to be seen whether it is possible to reduce this rate by the use of global properties. The results are summarized in Table 9.4.

Table 9.4: This table shows the change of the classification results in relation to Figure 9.9. The first column corresponds to episode 90 million, i. e., the last episode with 100% randomly selected actions ($\epsilon = 1.0$). The second column corresponds to episode 100 million, where a maxQ-policy is used ($\epsilon = 0.0$).

	random	maxQ	change
episodes	90 M	100 M	(rel.)
fail state	28.7%	17.9%	−37.7%
overtime/match	41.4%	57.8%	39.5%
overtime/miss	25.7%	8.5%	−67.0%
exact match	3.6%	15.3%	322.8%
no actions/match	0.4%	0.4%	−2.3%
no actions/miss	0.2%	0.2%	−9.1%
sum of positive results	45.4%	73.5%	61.8%

9.1.5 Interim Conclusion on Fusion with Reinforcement Learning

The conclusions arising from the results of this section can be summarized as follows:

- It is possible to increase the classification rate by a skillfully selection and combination of algorithms for local 3-D feature description. If all object classes are used, the improvement in comparison to the best single algorithm (FPFH with 9.4%) is more than 230%. If the reduced set of object classes is used, the classification rate still increases at least from 65% to 73.5%. However, it must be emphasized that the rate of exact matches increases from 0% to 15.3%.
- Trying to increase the classification rate using different limits during classification is not effective. This has the following reason: on the one hand the increase of the limit leads to a smaller number of class candidates, but on the other hand it results in a higher number of false negative results and thus in a higher rate of fail states.

9.2 Differentiation of the First State

Global parameters of the point cloud are required to differentiate the first state of the reinforcement learning framework. Therefore, Figure 9.10 illustrates the corresponding model again. The model differs from the model used until now only in the extension of the state with global properties of the point cloud.

These global properties are:

1. The number of keypoints determined for the point cloud.
This value is divided in 3 intervals, $[0, 37[$, $[38, 148[$, and $[149, \infty]$ as introduced in Section 7.2.3.
2. The ratios of the expansion of the point cloud along the principal axes.
This is done by the eigenvalues $e_1 \geq e_2 \geq e_3$ of the covariance matrix of the point cloud. The 4 possible states correspond to the cases where $e_1/e_2 \leq 3.0$ and $e_2/e_3 \leq 3.0$ are true or false, respectively.

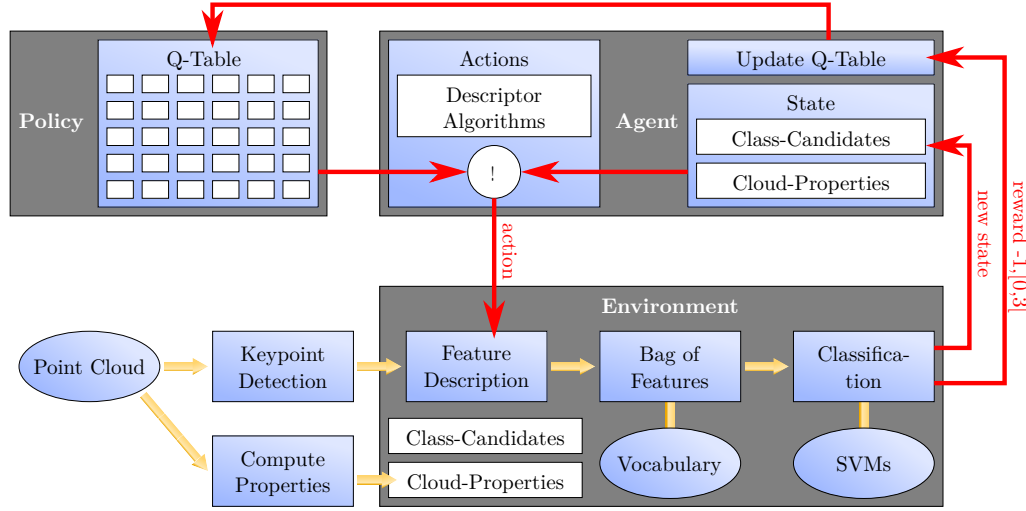


Figure 9.10: Model II: extending Model I by a differentiation of the first state. The previously used model is extended by global properties of the input point cloud. This enables the reinforcement learning framework to select different local 3-D feature description algorithms at the beginning.

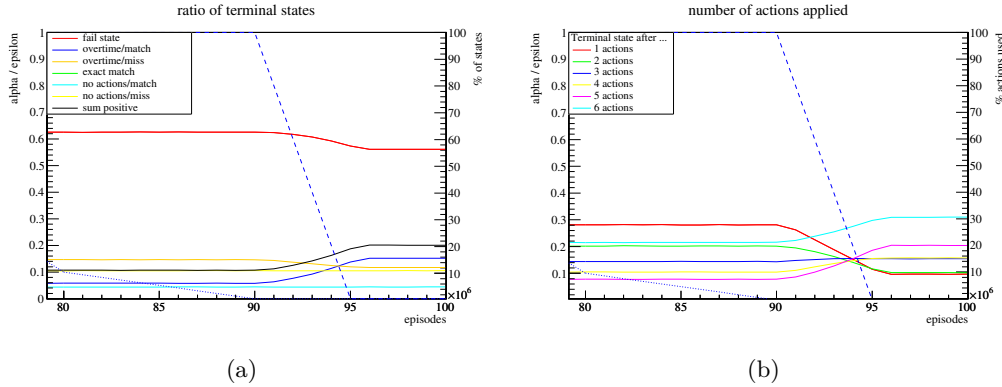


Figure 9.11: These two figures show the classification results when using global properties to distinguish the first state on all object classes.

9.2.1 Results

Following the previous sections, the graphs with the number of states and the course of the Q -values would be shown here. But since the shape of the curves don't change significantly, they are not relevant. For all object classes the final number of states is 2056005, while it is 3030 for 10 object classes.

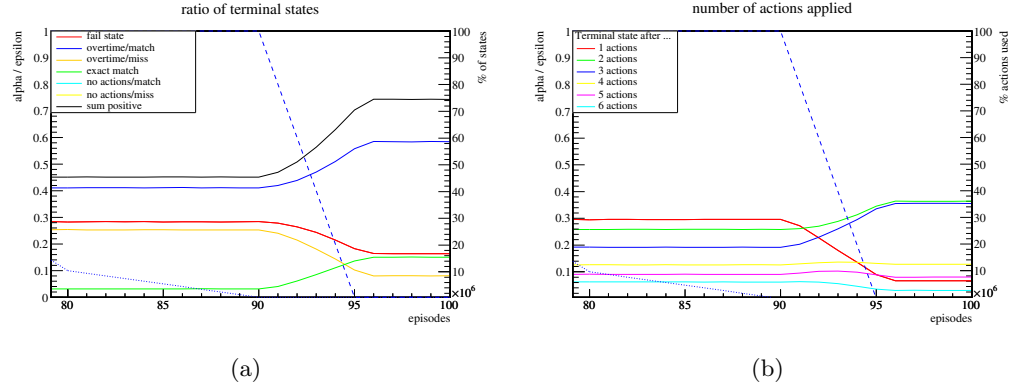


Figure 9.12: These two figures show the classification results when using global properties to distinguish the first state on 10 object classes.

More exciting, however, is the question of whether an improvement of the classification results can be achieved using these two simple global properties. Figure 9.11 shows the results for all object classes, while Table 9.5 contributes the corresponding values.

Table 9.5: Classification results when using global properties to distinguish the first state on all object classes.

	random	maxQ	change
episodes	90 M	100 M	(rel.)
fail state	62.8%	55.9%	−10.3%
overtime/match	6.3%	16.1%	147.9%
overtime/miss	15.1%	11.6%	−19.2%
exact match	0.0%	0.0%	0.0%
no actions/match	4.9%	5.4%	0.0%
no actions/miss	10.9%	11.0%	0.5%
sum of positive results	11.2%	21.5%	83.1%

At a first glance, it becomes clear that an improvement in the sum of positive results could not be achieved when using all 51 object classes (21.5% vs. 21.7%). The same applies to the fail states, where the results are completely identical.

On the other hand taking a look at Figure 9.12 of the results corresponding to 10 object classes, it seems that the approach tends to work.

Table 9.6: Classification results when using global properties to distinguish the first state on 10 object classes.

	random	maxQ	change
episodes	90 M	100 M	(rel.)
fail state	28.7%	16.3%	−41.5%
overtime/match	41.5%	58.6%	41.3%
overtime/miss	25.7%	8.5%	−66.6%
exact match	3.6%	16.0%	326.3%
no actions/match	0.4%	0.4%	−0.1%
no actions/miss	0.2%	0.2%	1.3%
sum of positive results	45.5%	75.0%	63.7%

The corresponding values of Table 9.6 show, that the number of fail states could be reduced from 17.9% to 16.3% while the number of exact matches could be increased from 15.3% to 16.0%. All in all the sum of positive results increased from 73.5% to 75.0%.

How strongly the learning process is influenced can be better explained with Table 9.7 and Table 9.8 shown below.

Table 9.7: Distribution of algorithms applied in each step of an episode without the influence of global properties.

step	3DSC	FPFH	PFH	SHOT	SI	USC	final state
1	0.00%	100.00%	0.00%	0.00%	0.00%	0.00%	13.90%
2	15.23%	0.00%	35.35%	10.02%	12.00%	13.51%	45.83%
3	12.53%	0.00%	9.51%	10.21%	3.43%	4.59%	26.98%
4	2.74%	0.00%	4.38%	1.44%	1.10%	3.63%	6.93%
5	0.64%	0.00%	2.69%	1.30%	1.47%	0.26%	3.31%
6	0.95%	0.00%	1.76%	0.04%	0.20%	0.11%	3.05%

Table 9.7 shows the distribution of algorithms that have been applied at each step of an episodes. Because no global properties have been used in this case, one can see that FPFH is always selected in the first step of each episode. The proportion of episodes, which are terminated after this first step is 13.9%.

The second Table 9.8 shows the same values with the use of global properties. On the one hand one can see that different algorithms will be used in first step

Table 9.8: Distribution of algorithms applied in each step of an episode with the influence of global properties.

step	3DSC	FPFH	PFH	SHOT	SI	USC	final state
1	0.00%	80.55%	0.00%	0.00%	1.02%	18.44%	6.16%
2	12.57%	5.02%	25.99%	21.25%	10.59%	18.42%	33.45%
3	21.10%	2.42%	14.25%	8.36%	11.20%	3.07%	38.38%
4	8.66%	2.71%	5.53%	1.56%	1.56%	1.98%	10.94%
5	0.88%	1.21%	6.23%	0.86%	0.71%	1.18%	8.17%
6	0.60%	0.09%	1.85%	0.13%	0.13%	0.08%	2.89%

of each episode, and on the other hand the proportion of episodes, which are terminated after this first step is with a value of 6.16% only half of the previous value. This is a quite satisfactory result, even if it is not reflected in the overall result.

9.2.2 Conclusion

The enhancement of the overall results that can be achieved using the global properties are quite small, but show that the method basically works. However, the significance of the simple global parameters used here is higher than expected. It could be shown that the fail states subsequent to the first applied algorithm could be more than halved using the global properties.

All in all, it must be assumed that better results can only be expected with a larger variety of local 3-D feature description algorithms and/or a more informative global description. The latter could be, for example, a very simple global descriptor, like the rotation invariant shape descriptor of Suzuki et al. [85], where they fit the point cloud into a unit cube, divide the cube into a coarse grid and count the points in each grid cell.

9.3 Adaptive Learning

As introduced in Section 7.2.4 adaptive learning means the ability to respond to future changes, without having to discard and relearn previously learned structures. In context of reinforcement learning this is typically done by means of an ϵ -greedy policy. As motivated in the same context in Section 7.2.4 an ϵ -greedy

policy does not make sense, because for unknown point clouds it can not be assessed if a classification was successful or not. Accordingly, during an episode no rewards can be propagated to randomly selected actions.

Therefore, the method is adapted as follows:

- With a probability of ϵ an episode called *random episode* is applied.
- During a random episode one of the known object with a known class is selected randomly. This enables the reinforcement learning framework to asses the classification result and to give an appropriate reward.
- During a random episode all actions will be selected randomly, too. This enables the reinforcement learning agent to select a previously unknown action from time to time.
- Otherwise, during a regular, non-random episode, any object can be classified. Here, the maxQ-policy is always used.
- Additionally, the Q -values of the Q -table will not be updated in regular episodes.

Subsequently, the reinforcement learning framework is initially trained with 5 of the 6 available algorithms on the dataset of 10 object classes. During an initial exploration phase both values α and ϵ are 1.0. After 25M episodes, i. e., after the Q -values have stabilized, the learning rate α is reduced to a value of 0.1. In this way, an eventual overfitting is compensated. At 45M episodes ϵ is also reduced to a value of 0.1. From this episode on the reinforcement learning is in an exploitation mode, even if 10% of the episodes are performed as random episodes. In this mode the reinforcement learning framework can be used for classification tasks of 3-D point clouds, even if they are new and their classes are previously unknown.

9.3.1 Results

To demonstrate the adaptability of the reinforcement learning framework, the respective missing algorithm is provided again with episode 50M. The courses of the classification curves as they are given in Figure 9.13 illustrate that the classification rates decrease a little bit first, while the Q -values are adjusted. Subsequently, the values increase in most cases to a higher value than before.

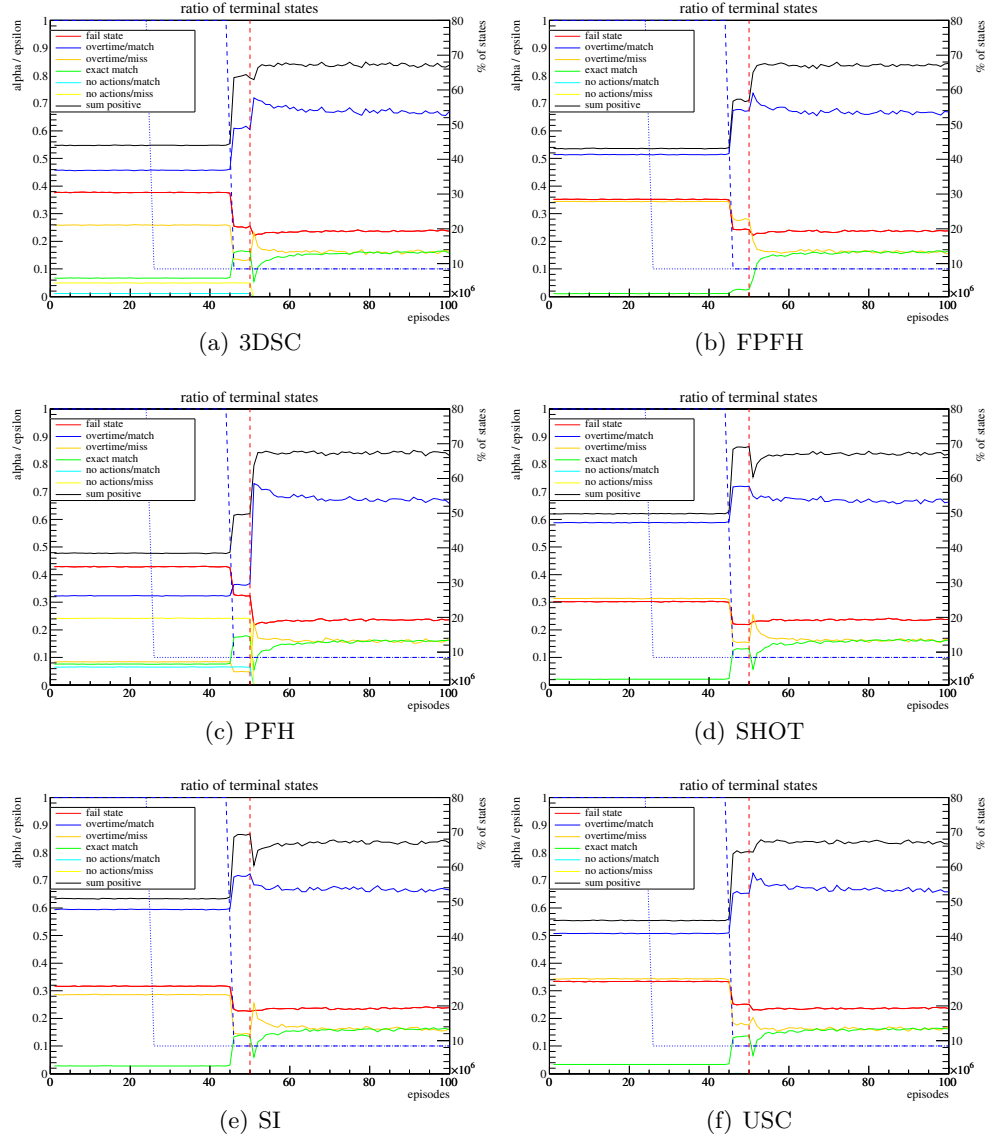


Figure 9.13: These 6 graphs illustrate the abilities of adaptive learning. The reinforcement learning environment starts with a pure exploration phase where α and ϵ have a value of 1.0, i.e., full learning rate with a completely randomly selection of actions. At 25M episodes the learning rate α is reduced from 1.0 to 0.1 (the blue dotted line), and at 45M episodes ϵ is reduced, from 1.0 to 0.1. From this moment the reinforcement learning framework does 90% exploitation and 10% exploration, i.e., 10% random episodes. Before reaching 50M episodes the agent has only 5 of 6 algorithm at his disposal. At 50M, marked with the red dashed line, the 6th algorithm which is specified under the respective graphs is added to the framework.

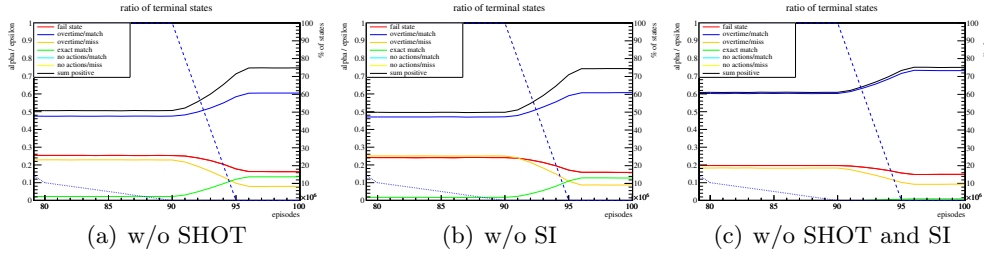


Figure 9.14: These three graphs show the classification results using the reinforcement learning framework without the algorithms SHOT and SI.

Upon closer examination of the graph two questions arise:

1. Why is the sum of the positive states (black line) despite the use of all 6 algorithms not as expected at a level of about 75%?
2. Why is the classification rate of the two cases where the algorithms SHOT and SI are initially excluded higher than with their use?

Both questions have the same cause: the random episodes. In a purely random selection of algorithms, a positive terminal state is achieved in 45.5% of all cases (see Table 9.6). The combination with the classification rate of the maxQ-policy results arithmetically in: $0.9 \cdot 75.0\% + 0.1 \cdot 45.5\% = 72.0\%$

Because of a learning rate of $\alpha = 0.1$ these values could not entirely be reached within the 100M episodes shown. However, a close look at the values shows that they change until the last episode shown.

If, in context of question two, one assumes that the two above-mentioned algorithms are "bad" for the learner, one would have to conclude that the overall results turn out better without the two algorithms. But this is not the case: Figure 9.14 reveals that the sum of positive terminal states keeps stable at approximately 75%. Instead, the exact matches drop slightly for the cases shown in Figure 9.14 (a) and (b) and reach a value close to zero in the third case shown in Figure 9.14 (c). Therefore, the random selection of the algorithms is also responsible for the behavior here. This implies that SHOT and SI will be used only in a particular order in relation to the other algorithms. However, this order will depend individually on the state of the reinforcement learning framework.

9.3.2 Conclusion

The analysis of the results clearly shows that the adaptive approach works very well. It shows that the classification rate converges very quickly even with a learning rate of $\alpha = 0.1$. Therefore in principle, a smaller value of ϵ is conceivable, so that the positive results classification can be maximized.

10

Comparative Evaluation of Results

The last three chapters have shown that it is possible to increase the classification results by a skillfully combination of several local 3-D feature description algorithms, without exceeding a defined time limit. The results show on the one hand how difficult a classification of noisy low-resolution 3-D point clouds is. But they also show that it is generally possible, and that proper classification rates can be reached for object classes which are to a certain degree disjoint. Therefore, the results are summarized in the following sections.

10.1 Initial Situation

The set of point clouds has been deliberately chosen to address the problems of a general realizable object classification in diverse environments. That means, that in addition to the technical characteristics such as the assignment of an object to an object class as listed in Section 7.3.1 some general characteristics of a data set influenced its choice:

- The data set should contain everyday objects.
- The objects should have been recorded by a 3-D capture device and should not have been created by a synthetic method such as CAD.
- Since in robotics low cost 3-D depth cameras, which operate on the principle of structured light have been used in recent years more frequently, it was also optimal that the point clouds of the chosen data set were created in the same manner.

However, the last item means that the point clouds have a relatively small size. With an average number of ≈ 5900 3-D points per object, the information content of the point cloud corresponds approximately to a 77×77 -pixel sized gray-scale image – but unstructured. So one has to make it clear that a classification task based solely on this small amount of information is a challenging task. An additional challenge is also the similarity of many object classes such as apple, onion, orange, peach, potato, and tomato.

For this reason, all experiments were carried out not only on the complete data set, which consists of 51 object classes, 300 distinct objects and 207481 low resolution depth scans (3-D point clouds) of different poses, but also on a reduced data set with 10 object classes and 53 distinct objects, whereby the latter still contains a considerable number of 37735 different 3-D point clouds.

10.2 Classification Results

To achieve a maximum classification rate for every single 3-D local feature description algorithms as well as for the reinforcement learning framework, the basic classification pipeline was analyzed and evaluated in a comprehensive examination of numerous parameters. Besides the optimization of various parameters of the entire classification pipeline, the classification rate of the support vector machines has been maximized. This step should especially reduce the number of false-negative results, i. e., the results that exclude an object from an object class, although the object belongs to the object class. Due to the design of the system this is crucial, since it is not possible to revise an exclusion.

The latter leads more or less inevitably to low precision values, i. e., a high number of false-positive results. In many cases, even when combining multiple algorithms, this does not lead to a unique assignment, but to a set of class candidate. In order to be still able to evaluate the result, the prediction values are summed up for each object class that remains in the set of class candidates. Finally, this sum is used to identify the object class with the highest sum of prediction values, which is then used for a mapping between the object and an object class. The latter leads to significantly weaker classification results of individual local 3-D feature descriptors in comparison to the results of $\approx 95\%$ reached during the training of the support vector machines, where only the cases

were taken into account in which an object could or couldn't be assigned to an object class.

The application of a single 3-D local feature description algorithm within the optimized basic classification pipeline lead to the following results: Exact classification results were neither possible with all object classes nor with the reduced set of 10 object classes. The algorithm, which assigns the most correct classes due to the prediction values of its support vector machines is in both cases the fast point feature histogram (FPFH) by Rusu et al. [71]. In case of all object classes a rate of 9.4% could be achieved, while a rate of 65.0% could be achieved for the reduced set of object classes.

Following the experiments of the basic classification pipeline the results of the reinforcement learning frameworks were determined. For this purpose, it was first examined whether an increase of the classification limit and thereby the prediction value of a support vector machine leads to better classification results. However, the analysis showed that an increase of the classification limit precisely had the opposite effect, because the increase of the classification limit led to an increased number of false-negative results, which should be avoided as noted above. For this reason, this approach was rejected.

But even without changing the classification limits it was possible to increase the rate of positive terminal states which would allow a correct assignment to the object class. While this classification rate for 10 object classes increased slightly from 65.0% for FPFH to 73.5% for the learned combination of different algorithms, the rate increased considerably from 9.4% to 21.7% for all object classes. Additionally, the rate of exactly assigned object classes could be increased from 0.0% to 15.3% and 4.9% respectively. This is a substantial improvement especially for the large data set.

Since the first state of the reinforcement learning framework in the approaches so far always was the state in which the set of class candidates consists of all object classes, the reinforcement learning agent was not able to select different algorithms in a first step. Therefore, a differentiation of the first state based on a few global properties of a point cloud was introduced. This approach was especially intended to reduce the relatively large number of fail states after the application of the first algorithm. Even if the reduction of the amount of fail states from 17.9% to 16.3% is relatively low, it has been shown that the basic principle works. In addition, it was shown that the amount of fail states after

the first applied algorithm could be more than halved using the global properties. This is quite a satisfactory result, even if it is not reflected in the overall result. In addition, the classification rate increased from 73.5% to 75% in the case of 10 object classes.

10.3 Adaptivity

One of the essential features of a reinforcement learning frameworks is the continuous adaptability to a changing environment. While other machine learning methods usually need to be retrained due to changes of the environment and the parameters, a reinforcement learning framework usually reacts on changes in an ongoing process, which affects the actual task of the reinforcement learning framework only slightly.

With an adaptive, randomized choice between ϵ -greedy and random episodes the proposed reinforcement learning framework demonstrated impressively how effectively this approach works. This enables the framework to successively add or remove local 3-D feature descriptors or to modify their parameters, e. g., the classification limits to observe the influence of the changes directly on the running system.

10.4 Computation Time

One topic that has not even been discussed yet is the computation time. In Section 7.3.2, the choice of the time limit was discussed, justified and specified to 10 seconds. An interesting final question is, how long does it really take to do a classification using this time limit and how many algorithms will be used within this time?

While in the case where global properties are not used each episode takes an average computation time of 4.4 seconds to apply 2.49 algorithms, the use of global properties increases the computation time slightly to 4.6 seconds. But within this time 2.9 algorithms are applied on average.

11

Conclusion and Outlook

11.1 Conclusion

This thesis has made novel contributions to the area of object classification based on 3-D point clouds. A reliable classification of objects is an essential milestone on the way to well-functioning scene understanding. However, object classification solely on the basis 2-D information, i.e., color images is often insufficient. Accordingly, in recent years much effort has gone into the development of local 3-D feature descriptions that allow for a robust classification. Nevertheless, a large-scale classification of objects based on 3-D point cloud data is still a major challenge.

This thesis could demonstrate how a reinforcement learning approach in the field of computer vision can improve the results of individual 3-D object classification approaches by learning strategies for a selection and successive application of different 3-D point cloud descriptors. It has been shown how a reinforcement learning framework can be put into position to find a good balance between the available time given and the classification abilities of individual algorithms. In concrete terms this means that the reinforcement learning framework learned to select individual sequences of local 3-D feature description algorithms, depending on the point cloud at hand, to get the best possible classification results. Furthermore, the usage of an on-line learning method such as the reinforcement learning method Q -learning provides a distinct advantage over off-line learning methods because of its flexibility under changing conditions. For example, it allows the adaptive integration of new algorithms into the classification process, while the system is on duty.

In summary, besides the described contributions to the field of machine learning this thesis has introduced a new concept to improve the classification rates in the area of 3-D object classification, of which a further investigation is considered to be desirable.

11.2 Future Work

While working on this thesis some new questions arose on different parts of the examined methods and models which are worth taking a closer look at. One of the first questions which arose was how the system scales using additional algorithms for local 3-D feature descriptions. This implies the question whether the classification rates continue to grow or stagnate at some point.

Furthermore, additional options for the selection of the first local 3-D feature description algorithm should be examined. As part of this thesis it was shown, that simple global properties of point clouds can be utilized to reduce the number of false classifications of the local 3-D feature description algorithm selected first. However, with the long-term aim of achieving a result close to 100% correct classifications, the errors caused by an unfavorable selection of the first algorithm have to be minimized. This could be done, for example, by using further global properties or by using additional global point cloud descriptors like the viewpoint feature histogram.

The investigations concerning the initial algorithm lead to a further question. Is there a potential correlation between specific global properties of a point cloud and the algorithms used at first? Or in other words: is it possible to identify certain characteristics of local 3-D feature description algorithms, which lead precisely to this differentiation? This information could possibly help to improve individual algorithms.

Finally, if it would be possible to achieve almost 100% correct classification results, the reinforcement learning framework could also be used to create new object classes, i.e., to autonomously learn the assignment of objects to classes it has not been exposed to before. Under the assumption that a classification is usually correct, it could also be assumed that an unsuccessful assignment of an object to a class means that this object does not belong to any of the known classes. If, in such a case, all available algorithms would be applied to the object and it would still be impossible to assign the object to an existing class, the object

would be regarded as a first instance of a new yet unnamed object class. In this way, new object classes could arise nearly automatically. In that case, new object classes have, of course, to be evaluated and labeled by a human observer from time to time.

A

Software

The following sections describe the software components, libraries, and the development environment used to develop all programs and tools.

A.1 System

Basis of all calculations are workstations from DELL with the following hardware specifications:

Parameter	Value
System	Dell Precision WorkStation T3500
CPU	Intel Xeon E5630 @2.53GHz
Memory	12GB DDR3 @1066MHz
GPU	NVIDIA GeForce GTX 670

The system used has the following configuration:

Parameter	Value
OS	Debian 8.0 (Jessie) GNU/Linux 64bit
Kernel	3.16.7 amd64
GNU C Library	2.19-18

A.2 Development Environment

The development environment consists of the following components:

Parameter	Value
Compiler	GNU Compiler Collection 4.8.4-1
Build Tools	GNU Make 4.0-8.1 CMake 3.0.2-1
IDE	KDevelop 4.7.0-1

A.3 Libraries

The following libraries are required to compile the programs:

Library	Note
libstdc++	GNU Standard C++ Library v3 4.8.4-1
OpenCV 3.0.0 *)	Open Source Computer Vision
PCL 1.8.0 *)	Point Cloud Library
ROOT 6.04/02 *)	Cern's modular scientific software toolkit
VTK v6.2.0 *)	The Visualization Toolkit

Note: all software tools marked with *) are compiled from source. Details can be found in the following sections.

A.3.1 VTK

The Visualization Toolkit (VTK) is an open-source, freely available software system for 3-D computer graphics, image processing, and visualization. It consists of a C++ class library and several interpreted interface layers including Tcl/Tk, Java, and Python. Version 6.2.0 of VTK (<http://www.vtk.org/>) is required to avoid a conflict with a simultaneous use of OpenCV and PCL, since the available VTK library included with the installed Debian version (libvtk5.8) is linked with Qt4, while Qt5 is required.

To compile the version used within this thesis (v6.2.0), the following commands can be used:

```
$ git clone https://gitlab.kitware.com/vtk/vtk.git
$ cd vtk
$ git checkout tags/v6.2.0
$ mkdir -p build
$ cd build
$ cmake \
    -DCMAKE_BUILD_TYPE=Release \
    -DCMAKE_CXX_COMPILER=/usr/bin/g++-4.8 \
    -DCMAKE_C_COMPILER=/usr/bin/gcc-4.8 \
    -DVTK_QT_VERSION:STRING=5 \
    -DVTK_Group_Qt=ON \
    ..
$ make -j4
$ make install
$ ldconfig
```

A.3.2 OpenCV

OpenCV (Open Source Computer Vision Library) available at <http://opencv.org> is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms.

To compile the version used within this thesis (3.0.0), the following commands can be used:

```
$ git clone https://github.com/Itseez/opencv.git
$ cd opencv
$ git checkout tags/3.0.0
$ mkdir -p build
$ cd build
$ cmake -Wno-dev \
    -DCMAKE_BUILD_TYPE=Release \
    -DCMAKE_CXX_COMPILER=/usr/bin/g++-4.8 \
    -DCMAKE_C_COMPILER=/usr/bin/gcc-4.8 \
    -DBUILD_DOCS=On \
    -DBUILD_EXAMPLES=On \
    -DWITH_OPENGL=On \
    -DWITH_QT=5 \
    -DWITH_VTK=On \
    -DWITH_OPENMP=On \
    ..
$ make -j4 && make doxygen && make install && ldconfig
```

A.3.3 PCL

The Point Cloud Library (<http://pointclouds.org>) is a large open source library for 3-D point cloud processing.

To compile the version used within this thesis (1.8.0), the following commands can be used:

```
$ git clone https://github.com/PointCloudLibrary/pcl.git
$ cd pcl
$ mkdir -p build
$ cd build
$ cmake -Wno-dev \
    -DCMAKE_BUILD_TYPE=Release \
    -DCMAKE_CXX_COMPILER=/usr/bin/g++-4.8 \
```

```
-DCMAKE_C_COMPILER=/usr/bin/gcc-4.8 \  
-DBUILD_CUDA=ON \  
-DBUILD_cuda_apps=ON \  
-DBUILD_cuda_io=ON \  
-DBUILD_GPU=ON \  
-DBUILD_apps=ON \  
-DBUILD_apps_cloud_composer=ON \  
-DBUILD_apps_modeler=ON \  
-DBUILD_apps_point_cloud_editor=ON \  
-DBUILD_examples=ON \  
-DWITH_DOCS=ON \  
-DWITH_TUTORIALS=ON \  
..  
$ make -j3 && make install && ldconfig
```

A.3.4 ROOT

ROOT is a framework for data processing, developed at CERN. The ROOT system (<http://root.cern.ch/>) is used to handle, store and analyze the data. The experiments store both their raw data and intermediate, processed results using ROOT. Especially, all of the graphs and histograms in this thesis are created with ROOT.

To compile the version used within this thesis (v6.04/02), the following commands can be used:

```
$ git clone http://root.cern.ch/git/root.git  
$ cd root  
$ git checkout tags/v6-04-02  
$ ./configure --prefix=/usr/local  
$ make -j3 && make install && ldconfig && make install  
$ ldconfig
```

B

Datasets

This appendix provides an overview of the data sets used.

B.1 The Stanford 3-D Scanning Repository

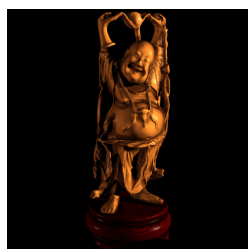
The Stanford 3-D Scanning Repository is a set of computer graphics test models for 3-D graphics. The data was collected by using a technique called range scan that was developed by Greg Turk of Georgia Institute of Technology and Marc Levoy of Stanford University. It consists of the following objects:



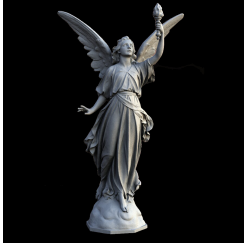
filename:	bun_zipper.ply			
description:	'Stanford Bunny' in 4 resolutions:			
number of vertices:	35947	8171	1889	453
unique vertices:	35947	8171	1889	453
number of faces:	69451	16301	3851	948
unique faces:	69451	16214	3768	908
unique edges:	104288	24363	5661	1363



filename:	dragon_vrip.ply			
description:	'Dragon' in 4 resolutions:			
number of vertices:	437645	100250	22998	5205
unique vertices:	435545	100250	22998	5205
number of faces:	871414	202520	47794	11102
unique faces:	871414	201031	46540	10600
unique edges:	1307004	301207	69509	15796



filename:	happy_vrip.ply			
description:	'Happy Buddha' in 4 resolutions:			
number of vertices:	543652	144647	32328	7108
unique vertices:	543524	144647	32328	7108
number of faces:	1087716	293232	67240	15536
unique faces:	1087693	290633	65590	14765
unique edges:	1631286	435252	97945	21852



filename:	lucy.ply
description:	Lucy
number of vertices:	14027872
unique vertices:	14027870
number of faces:	28055742
unique faces:	28055728
unique edges:	42083637

These models are used to approximate the mesh resolution.

B.2 Dataset from the 3-D Shape Retrieval Contest 2013

The objects of the following dataset from the “3-D shape Retrieval Contest 2013” [76] were captured with the Microsoft Kinect camera. In addition to the Stanford models mentioned above, these models are also used to approximate the mesh resolution.



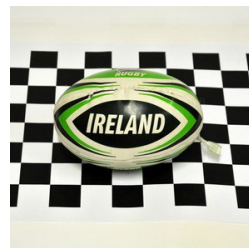
filename: 22.ply
description: teddy bear
number of vertices: 71403
unique vertices: 11903
number of faces: 23808
unique faces: 23808
unique edges: 35712



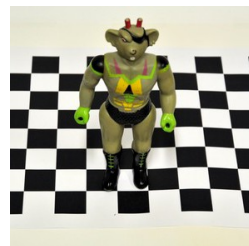
filename: 29.ply
description: finger puppet
number of vertices: 126734
unique vertices: 21128
number of faces: 42260
unique faces: 42260
unique edges: 63390



filename: 49.ply
description: clock
number of vertices: 207990
unique vertices: 34670
number of faces: 69348
unique faces: 69348
unique edges: 104022



filename: 56.ply
description: rugby ball
number of vertices: 144563
unique vertices: 24112
number of faces: 48220
unique faces: 48220
unique edges: 72330



filename: 57.ply
description: alien rat
number of vertices: 105383
unique vertices: 17572
number of faces: 35140
unique faces: 35140
unique edges: 52710



filename: 141.ply
 description: cow
 number of vertices: 67910
 unique vertices: 11322
 number of faces: 22640
 unique faces: 22640
 unique edges: 33960



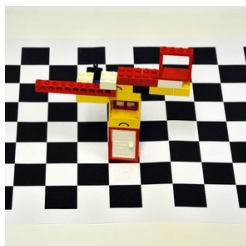
filename: 144.ply
 description: santa claus
 number of vertices: 139024
 unique vertices: 23177
 number of faces: 46366
 unique faces: 46366
 unique edges: 69549



filename: 146.ply
 description: wooden air plane
 number of vertices: 129027
 unique vertices: 21496
 number of faces: 43020
 unique faces: 43020
 unique edges: 64530



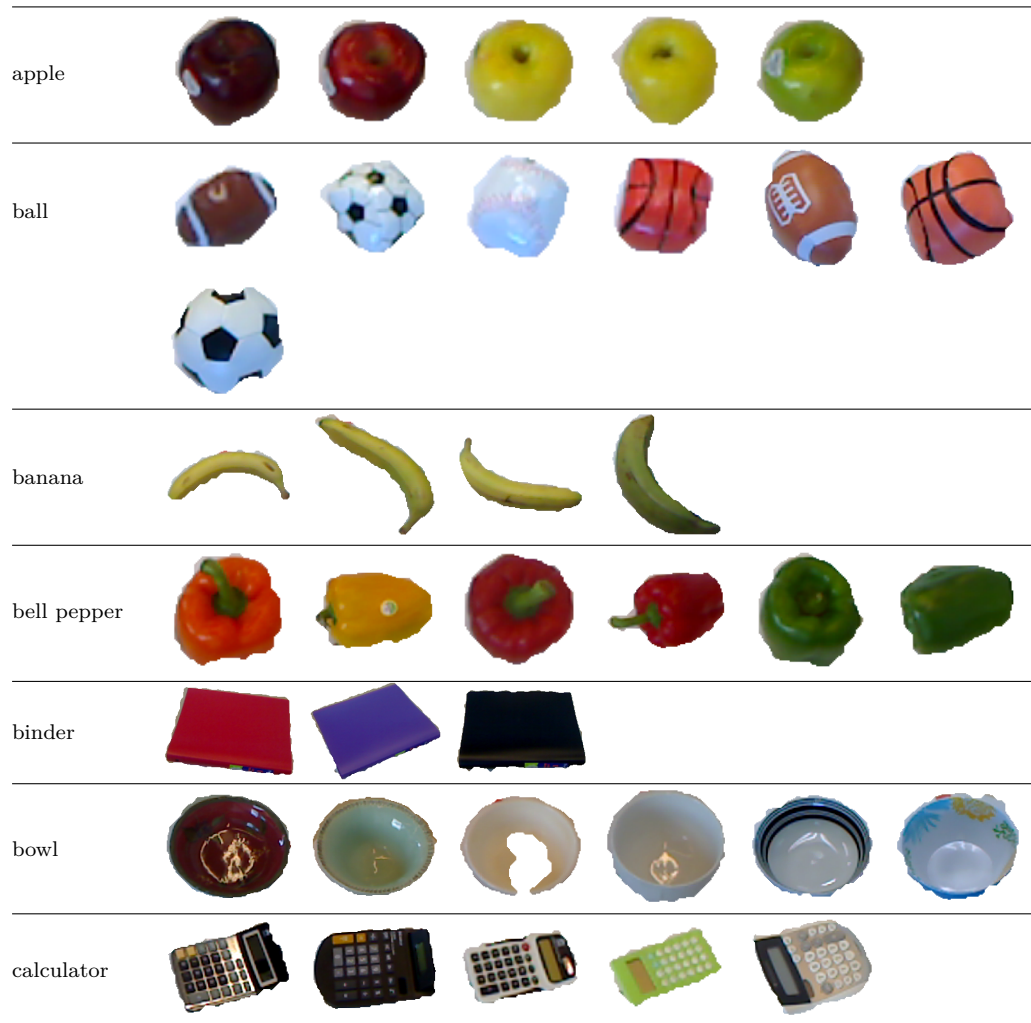
filename: 205.ply
 description: castle clock
 number of vertices: 92920
 unique vertices: 15491
 number of faces: 30978
 unique faces: 30978
 unique edges: 46467



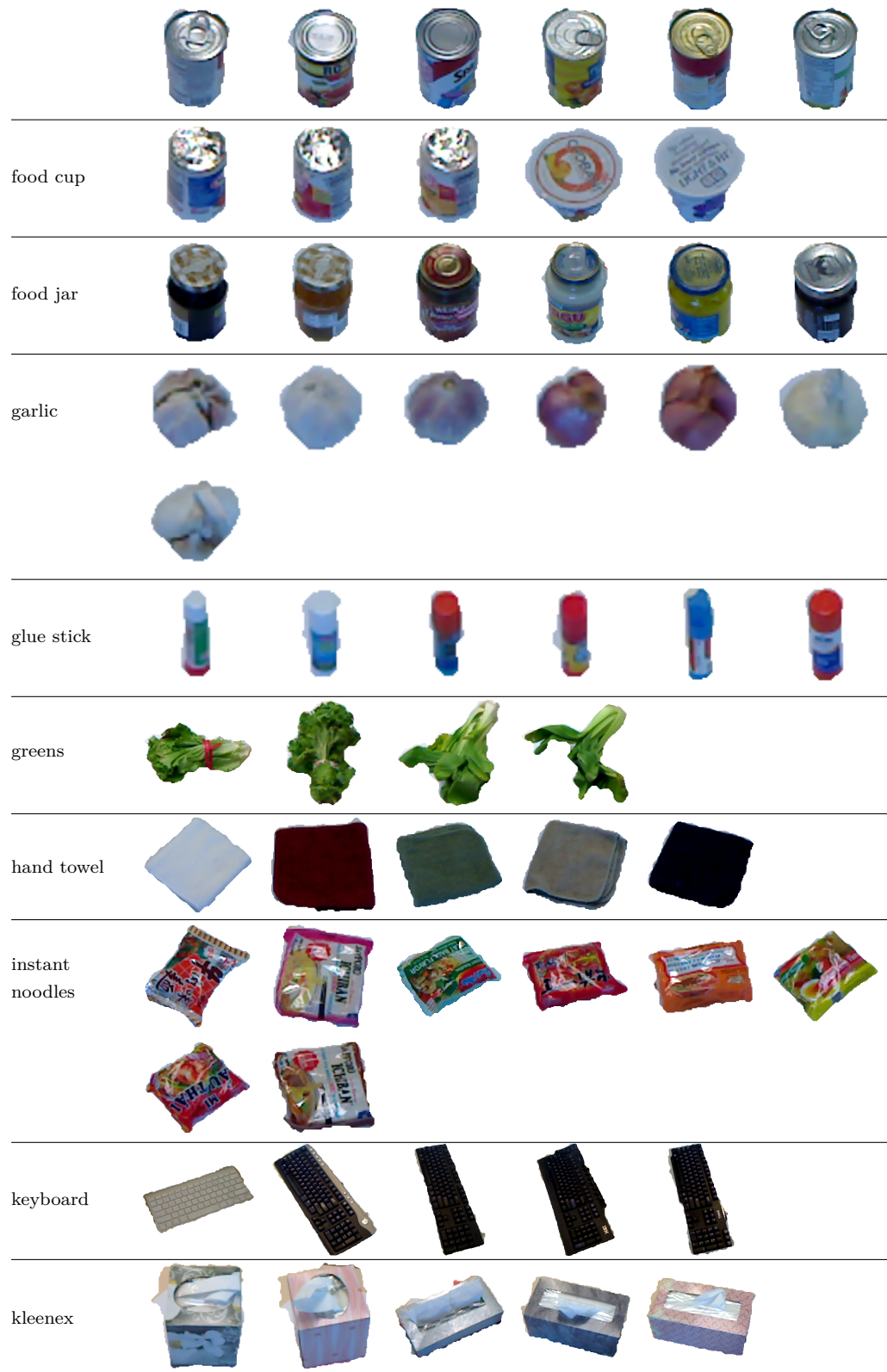
filename: 210.ply
 description: LEGO crane
 number of vertices: 64978
 unique vertices: 10835
 number of faces: 21670
 unique faces: 21670
 unique edges: 32505

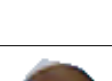
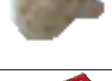
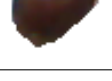
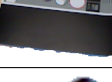
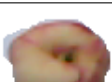
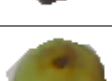
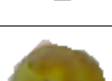
B.3 RGB-D Object Dataset from the University of Washington

The dataset contains 300 objects in 51 categories. Cropped photos from the objects that were used within each of the 51 categories are shown on the next pages. The crop masks used in these images correspond to the 3-D data that was used in this thesis. The latter is characterized by the fact that areas covered by glossy reflections are missing. In the same areas also no depth values could be determined due to the reflection. In addition, it can be seen at objects with thin parts such as pliers or scissors that also the x - and y -resolution of 3-D cameras that work with structured light is limited.



camera	
cap	
cell phone	
cereal box	
coffee mug	
comb	
dry battery	
flashlight	
food bag	
food box	
food can	



lemon						
lightbulb						
lime						
marker						
						
mushroom						
notebook						
onion						
orange						
peach						
pear						
pitcher						





toothbrush



toothpaste



water bottle

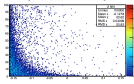
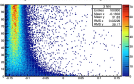
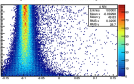
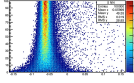
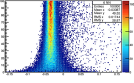
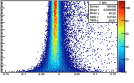
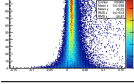
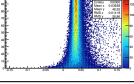
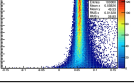


This chapter provides the full results of those experiments and evaluations, which could be given only in extracts in the evaluation.

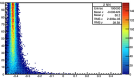
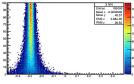
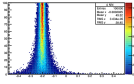
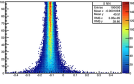
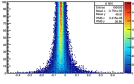
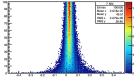
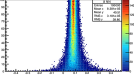
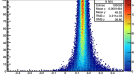
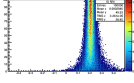
C.1 Approximation of the Mesh Resolution

The figures below show all histograms of differences between the mesh resolution and the mean Euclidean distances calculated for n nearest neighbors. In each histogram the x-axis shows the difference to the corresponding mesh resolution and the y-axis shows the number of randomly selected points which have been used to calculate the mean point distance to the n nearest neighbors.

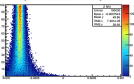
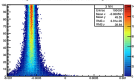
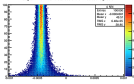
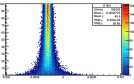
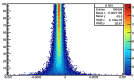
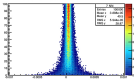
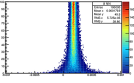
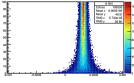
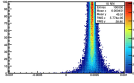
The tables show all results of approximations of mesh resolutions for a sample size of 50 randomly selected 3D points. The *n.n.* column show the number nearest neighbors used to calculate the mean. The columns *app. mean* and *mean diff.* show the approximation results and the differences to the mesh resolution based on edge lengths. The column *diff. (%)* is the mean difference divided by the mesh resolution. The highlighted rows show the results with the smallest difference between the mesh resolution and the approximated values.

'Lucy' in full resolution						
	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	0.35137881	-0.13625415	± 0.00620694	0.03876232	-38.78
	3	0.35517195	-0.13246101	± 0.00063321	0.01964994	-37.29
	4	0.39458594	-0.09304702	± 0.00059109	0.01869189	-23.58
	5	0.42821923	-0.05941373	± 0.00057295	0.01811828	-13.87
	6	0.45372069	-0.03391227	± 0.00040262	0.01270632	-7.47
	7	0.47769997	-0.00993299	± 0.00039771	0.01255136	-2.08
	8	0.50063982	0.01300686	± 0.00036343	0.01146380	2.60
	9	0.52433377	0.03670081	± 0.00033833	0.01066691	7.00
	10	0.54622912	0.05859617	± 0.00034288	0.01081029	10.73

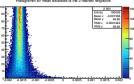
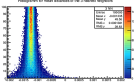
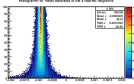
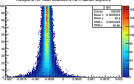
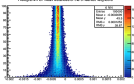
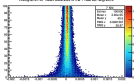
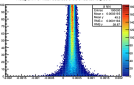
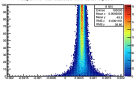
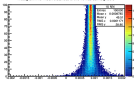
'Stanford Bunny' in full resolution (resolution 1)

	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	0.00104699	-0.00042363	± 0.00000061	0.00001940	-40.46
	3	0.00118107	-0.00028955	± 0.00000058	0.00001843	-24.52
	4	0.00126949	-0.00020113	± 0.00000059	0.00001868	-15.84
	5	0.00136226	-0.00010836	± 0.00000058	0.00001833	-7.95
	6	0.00143414	-0.00003648	± 0.00000057	0.00001813	-2.54
	7	0.00149832	0.00002769	± 0.00000059	0.00001860	1.85
	8	0.00155601	0.00008539	± 0.00000061	0.00001934	5.49
	9	0.00161863	0.00014801	± 0.00000062	0.00001968	9.14
	10	0.00167703	0.00020641	± 0.00000064	0.00002029	12.31

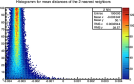
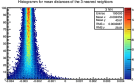
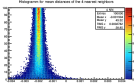
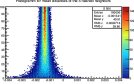
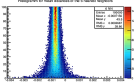
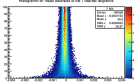
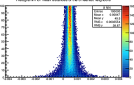
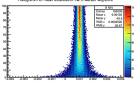
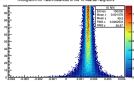
'Stanford Bunny' in reduced resolution (resolution 2)

	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	0.00229368	-0.00075488	± 0.00000159	0.00005030	-32.91
	3	0.00245835	-0.00059020	± 0.00000143	0.00004510	-24.01
	4	0.00261149	-0.00043707	± 0.00000132	0.00004162	-16.74
	5	0.00277767	-0.00027089	± 0.00000129	0.00004091	-9.75
	6	0.00293929	-0.00010926	± 0.00000124	0.00003933	-3.72
	7	0.00308892	0.00004036	± 0.00000118	0.00003743	1.31
	8	0.00322700	0.00017844	± 0.00000114	0.00003604	5.53
	9	0.00336415	0.00031559	± 0.00000114	0.00003617	9.38
	10	0.00350144	0.00045288	± 0.00000117	0.00003694	12.93

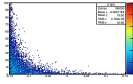
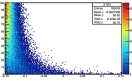
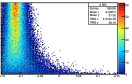
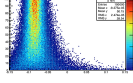
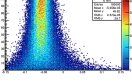
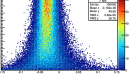
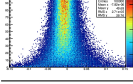
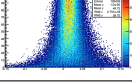
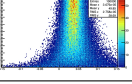
'Stanford Bunny' in reduced resolution (resolution 3)

	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	0.00470970	-0.00157311	± 0.00000318	0.00010063	-33.40
	3	0.00504578	-0.00123703	± 0.00000282	0.00008930	-24.52
	4	0.00536204	-0.00092077	± 0.00000256	0.00008089	-17.17
	5	0.00569007	-0.00059274	± 0.00000249	0.00007871	-10.42
	6	0.00601250	-0.00027031	± 0.00000243	0.00007673	-4.50
	7	0.00631754	0.00003474	± 0.00000232	0.00007326	0.55
	8	0.00660285	0.00032004	± 0.00000226	0.00007155	4.85
	9	0.00688427	0.00060146	± 0.00000228	0.00007219	8.74
	10	0.00716248	0.00087967	± 0.00000232	0.00007334	12.28

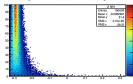
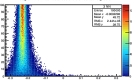
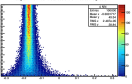
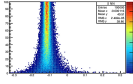
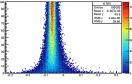
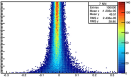
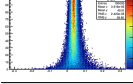
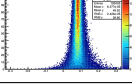
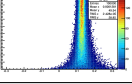
'Stanford Bunny' in reduced resolution (resolution 4)

	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	0.00923791	-0.00336723	± 0.00000671	0.00021233	-36.45
	3	0.00993927	-0.00266586	± 0.00000611	0.00019335	-26.82
	4	0.01060562	-0.00199952	± 0.00000572	0.00018086	-18.85
	5	0.01126143	-0.00134370	± 0.00000551	0.00017411	-11.93
	6	0.01188754	-0.00071759	± 0.00000540	0.00017065	-6.04
	7	0.01248901	-0.00011612	± 0.00000526	0.00016623	-0.93
	8	0.01307578	0.00047065	± 0.00000505	0.00015977	3.60
	9	0.01363637	0.00103124	± 0.00000494	0.00015633	7.56
	10	0.01418421	0.00157907	± 0.00000492	0.00015571	11.13

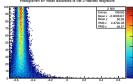
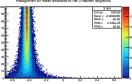
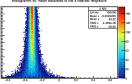
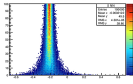
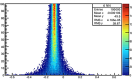
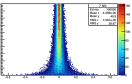
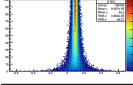
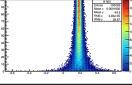
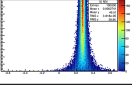
'Dragon' in full resolution (resolution 1)

	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	0.00031057	-0.00014285	± 0.00000171	0.00000640	-46.00
	3	0.00031809	-0.00013533	± 0.00000049	0.00001170	-42.54
	4	0.00034077	-0.00011265	± 0.00000053	0.00001684	-33.06
	5	0.00036981	-0.00008362	± 0.00000055	0.00001726	-22.61
	6	0.00039672	-0.00005671	± 0.00000056	0.00001764	-14.29
	7	0.00042213	-0.00003129	± 0.00000057	0.00001813	-7.41
	8	0.00044624	-0.00000719	± 0.00000059	0.00001864	-1.61
	9	0.00046935	0.00001592	± 0.00000060	0.00001912	3.39
	10	0.00049155	0.00003813	± 0.00000062	0.00001962	7.76

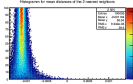
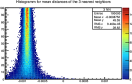
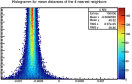
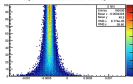
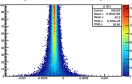
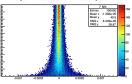
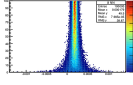
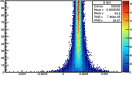
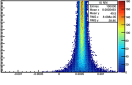
'Dragon' in reduced resolution (resolution 2)

	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	0.00069644	-0.00029282	± 0.00000053	0.00001670	-42.05
	3	0.00075913	-0.00023014	± 0.00000050	0.00001594	-30.32
	4	0.00081769	-0.00017158	± 0.00000049	0.00001553	-20.98
	5	0.00087362	-0.00011564	± 0.00000049	0.00001540	-13.24
	6	0.00092701	-0.00006225	± 0.00000048	0.00001525	-6.72
	7	0.00097762	-0.00001164	± 0.00000048	0.00001513	-1.19
	8	0.00102583	0.00003656	± 0.00000048	0.00001518	3.56
	9	0.00107247	0.00008320	± 0.00000048	0.00001534	7.76
	10	0.00111782	0.00012856	± 0.00000049	0.00001554	11.50

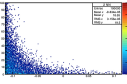
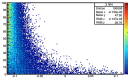
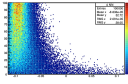
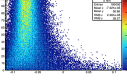
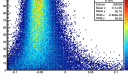
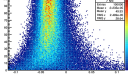
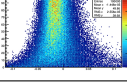
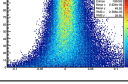
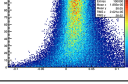
'Dragon' in reduced resolution (resolution 3)

	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	0.00150494	-0.00054663	± 0.00000104	0.00003304	-36.32
	3	0.00162169	-0.00042988	± 0.00000093	0.00002941	-26.51
	4	0.00173019	-0.00032137	± 0.00000086	0.00002730	-18.57
	5	0.00183875	-0.00021281	± 0.00000084	0.00002663	-11.57
	6	0.00194520	-0.00010637	± 0.00000082	0.00002589	-5.47
	7	0.00204684	-0.00000473	± 0.00000078	0.00002476	-0.23
	8	0.00214213	0.00009057	± 0.00000076	0.00002415	4.23
	9	0.00223468	0.00018311	± 0.00000076	0.00002408	8.19
	10	0.00232518	0.00027361	± 0.00000077	0.00002431	11.77

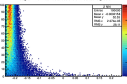
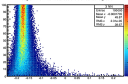
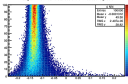
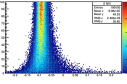
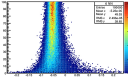
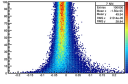
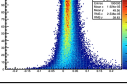
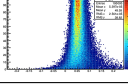
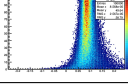
'Dragon' in reduced resolution (resolution 4)

	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	0.00311774	-0.00111048	± 0.00000219	0.00006915	-35.62
	3	0.00335064	-0.00087757	± 0.00000194	0.00006126	-26.19
	4	0.00357117	-0.00065704	± 0.00000177	0.00005592	-18.40
	5	0.00379464	-0.00043357	± 0.00000171	0.00005414	-11.43
	6	0.00401105	-0.00021716	± 0.00000167	0.00005278	-5.41
	7	0.00421530	-0.00001291	± 0.00000160	0.00005074	-0.31
	8	0.00440744	0.00017923	± 0.00000156	0.00004927	4.07
	9	0.00459447	0.00036626	± 0.00000156	0.00004941	7.97
	10	0.00477788	0.00054966	± 0.00000160	0.00005056	11.50

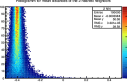
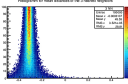
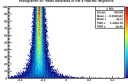
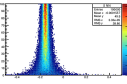
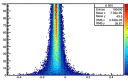
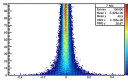
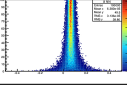
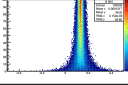
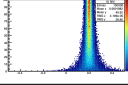
'Happy Buddha' in full resolution (resolution 1)

	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	0.00023908	-0.00010609	± 0.00000282	0.00000846	-44.37
	3	0.00024274	-0.00010244	± 0.00000048	0.00000951	-42.20
	4	0.00025438	-0.00009079	± 0.00000049	0.00001489	-35.69
	5	0.00027373	-0.00007144	± 0.00000055	0.00001729	-26.10
	6	0.00029401	-0.00005116	± 0.00000057	0.00001811	-17.40
	7	0.00031331	-0.00003186	± 0.00000060	0.00001884	-10.17
	8	0.00033166	-0.00001351	± 0.00000062	0.00001955	-4.07
	9	0.00034918	0.00000401	± 0.00000064	0.00002023	1.15
	10	0.00036598	0.00002080	± 0.00000066	0.00002091	5.68

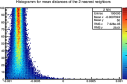
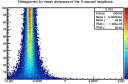
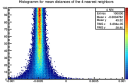
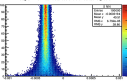
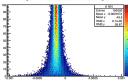
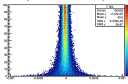
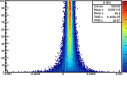
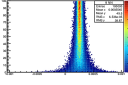
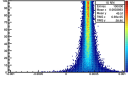
'Happy Buddha' in reduced resolution (resolution 2)

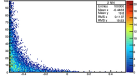
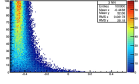
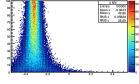
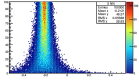
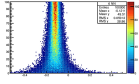
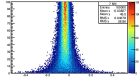
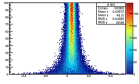
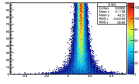
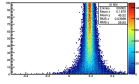
	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	0.00050599	-0.00021803	± 0.00000045	0.00001368	-43.09
	3	0.00055032	-0.00017369	± 0.00000049	0.00001546	-31.56
	4	0.00059319	-0.00013082	± 0.00000050	0.00001566	-22.05
	5	0.00063382	-0.00009020	± 0.00000051	0.00001608	-14.23
	6	0.00067256	-0.00005146	± 0.00000052	0.00001650	-7.65
	7	0.00070928	-0.00001473	± 0.00000054	0.00001698	-2.08
	8	0.00074425	0.00002023	± 0.00000055	0.00001752	2.72
	9	0.00077813	0.00005412	± 0.00000057	0.00001806	6.96
	10	0.00081107	0.00008705	± 0.00000059	0.00001861	10.73

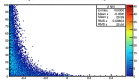
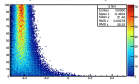
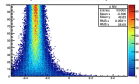
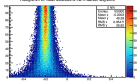
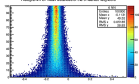
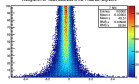
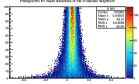
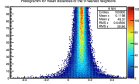
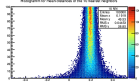
'Happy Buddha' in reduced resolution (resolution 3)

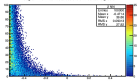
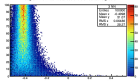
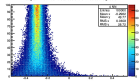
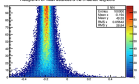
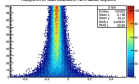
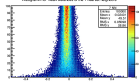
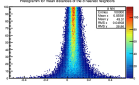
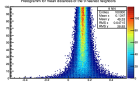
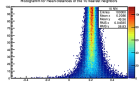
	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	0.00111400	-0.00040115	± 0.00000079	0.00002498	-36.01
	3	0.00120051	-0.00031464	± 0.00000072	0.00002285	-26.21
	4	0.00128028	-0.00023487	± 0.00000068	0.00002164	-18.35
	5	0.00135871	-0.00015643	± 0.00000066	0.00002099	-11.51
	6	0.00143529	-0.00007986	± 0.00000065	0.00002053	-5.56
	7	0.00150868	-0.00000646	± 0.00000063	0.00002004	-0.43
	8	0.00157839	0.00006325	± 0.00000063	0.00001987	4.01
	9	0.00164632	0.00013118	± 0.00000064	0.00002009	7.97
	10	0.00171296	0.00019781	± 0.00000065	0.00002041	11.55

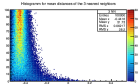
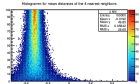
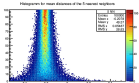
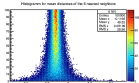
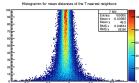
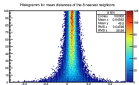
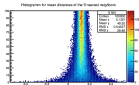
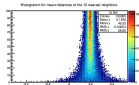
'Happy Buddha' in reduced resolution (resolution 4)

	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	0.00232557	-0.00080491	± 0.00000159	0.00005036	-34.61
	3	0.00249375	-0.00063674	± 0.00000141	0.00004468	-25.53
	4	0.00265049	-0.00047999	± 0.00000129	0.00004085	-18.11
	5	0.00280961	-0.00032087	± 0.00000126	0.00003986	-11.42
	6	0.00296176	-0.00016873	± 0.00000124	0.00003915	-5.70
	7	0.00310749	-0.00002299	± 0.00000122	0.00003844	-0.74
	8	0.00324530	0.00011482	± 0.00000120	0.00003781	3.54
	9	0.00338383	0.00025335	± 0.00000123	0.00003900	7.49
	10	0.00351897	0.00038848	± 0.00000128	0.00004055	11.04

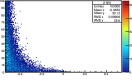
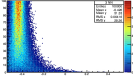
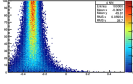
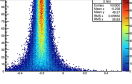
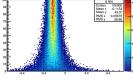
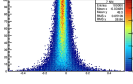
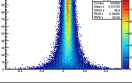
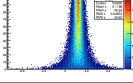
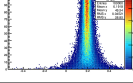
SHREC'13 object 22						
	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	1.09857595	-0.51206929	± 0.00383148	0.02453344	-46.61
	3	1.15788835	-0.45275689	± 0.00145183	0.04488957	-39.10
	4	1.28734788	-0.32329736	± 0.00125921	0.03981984	-25.11
	5	1.39938945	-0.21125578	± 0.00108751	0.03439017	-15.10
	6	1.49075900	-0.11988623	± 0.00097743	0.03090919	-8.04
	7	1.57352661	-0.03711863	± 0.00090484	0.02861369	-2.36
	8	1.65140697	0.04076174	± 0.00085199	0.02694224	2.47
	9	1.72742225	0.11677702	± 0.00081681	0.02582983	6.76
	10	1.79912839	0.18848315	± 0.00079355	0.02509431	10.48

SHREC'13 object 29						
	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	1.09087494	-0.50806699	± 0.00251479	0.02502181	-46.57
	3	1.16974167	-0.42920025	± 0.00144296	0.04526382	-36.69
	4	1.29046851	-0.30847342	± 0.00124660	0.03942090	-23.90
	5	1.39761496	-0.20132697	± 0.00110836	0.03504949	-14.41
	6	1.48519538	-0.11374654	± 0.00102160	0.03230577	-7.66
	7	1.56539965	-0.03354228	± 0.00095677	0.03025567	-2.14
	8	1.64167482	0.04273289	± 0.00090659	0.02866880	2.60
	9	1.71851877	0.11957684	± 0.00088180	0.02788494	6.96
	10	1.79091594	0.19197402	± 0.00086694	0.02741498	10.72

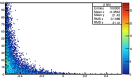
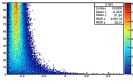
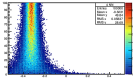
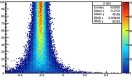
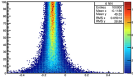
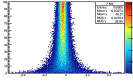
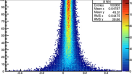
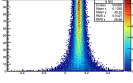
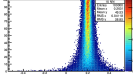
SHREC'13 object 49						
	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	1.11304506	-0.50650715	± 0.00219400	0.02877403	-45.51
	3	1.20232242	-0.41722979	± 0.00150374	0.04740949	-34.70
	4	1.32206149	-0.29749072	± 0.00127658	0.04036889	-22.50
	5	1.42579193	-0.19376028	± 0.00113422	0.03586719	-13.59
	6	1.51214879	-0.10740342	± 0.00106044	0.03353415	-7.10
	7	1.59634297	-0.02320924	± 0.00101488	0.03209331	-1.45
	8	1.67625568	0.05670347	± 0.00098079	0.03101521	3.38
	9	1.75530689	0.13575467	± 0.00096144	0.03040352	7.73
	10	1.82959042	0.21003821	± 0.00094980	0.03003521	11.48

SHREC'13 object 56						
	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	1.11131837	-0.51756655	± 0.00436794	0.02796849	-46.57
	3	1.17909399	-0.44979094	± 0.00140886	0.04378810	-38.15
	4	1.30758114	-0.32130379	± 0.00125779	0.03977479	-24.57
	5	1.42097886	-0.20790606	± 0.00111112	0.03513680	-14.63
	6	1.51268232	-0.11620260	± 0.00102263	0.03233830	-7.68
	7	1.59552272	-0.03336221	± 0.00096106	0.03039137	-2.09
	8	1.67340803	0.04452311	± 0.00091293	0.02886928	2.66
	9	1.75195075	0.12306583	± 0.00088954	0.02812962	7.02
	10	1.82526099	0.19637606	± 0.00087548	0.02768513	10.76

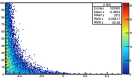
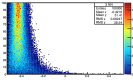
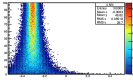
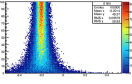
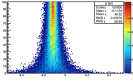
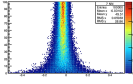
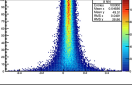
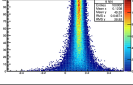
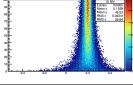
SHREC'13 object 57

	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	1.10649059	-0.51696524	± 0.00257846	0.02172647	-46.72
	3	1.18631620	-0.43713963	± 0.00137661	0.04307260	-36.85
	4	1.30862864	-0.31482719	± 0.00121328	0.03836734	-24.06
	5	1.41734444	-0.20611139	± 0.00108048	0.03416764	-14.54
	6	1.50576873	-0.11768710	± 0.00099913	0.03159514	-7.82
	7	1.58644845	-0.03700738	± 0.00094638	0.02992731	-2.33
	8	1.66332667	0.03987084	± 0.00090344	0.02856938	2.40
	9	1.74048209	0.11702626	± 0.00088134	0.02787032	6.72
	10	1.81363690	0.19018107	± 0.00086914	0.02748452	10.49

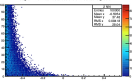
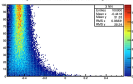
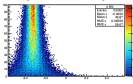
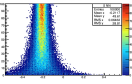
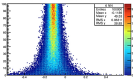
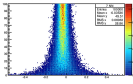
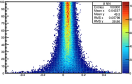
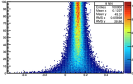
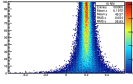
SHREC'13 object 141

	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	1.10842467	-0.51649310	± 0.00260938	0.01629555	-46.60
	3	1.17230545	-0.45261232	± 0.00142741	0.04418067	-38.61
	4	1.29930401	-0.32561376	± 0.00127353	0.04027254	-25.06
	5	1.41189311	-0.21302466	± 0.00110857	0.03505619	-15.09
	6	1.50570396	-0.11921381	± 0.00100312	0.03172154	-7.92
	7	1.59172876	-0.03318901	± 0.00093044	0.02942320	-2.09
	8	1.67249761	0.04757984	± 0.00088021	0.02783460	2.84
	9	1.75097263	0.12605486	± 0.00084711	0.02678790	7.20
	10	1.82496473	0.20004696	± 0.00082381	0.02605125	10.96

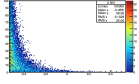
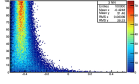
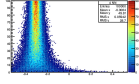
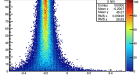
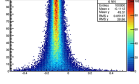
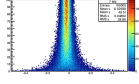
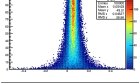
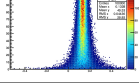
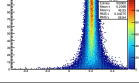
SHREC'13 object 144

	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	1.09090189	-0.50703346	± 0.00218180	0.02115329	-46.48
	3	1.16660151	-0.43133384	± 0.00145638	0.04559202	-36.97
	4	1.28763404	-0.31030131	± 0.00128611	0.04067025	-24.10
	5	1.39412711	-0.20380824	± 0.00114103	0.03608260	-14.62
	6	1.48245410	-0.11548125	± 0.00105712	0.03342901	-7.79
	7	1.56492910	-0.03300625	± 0.00100340	0.03173018	-2.11
	8	1.64288338	0.04494803	± 0.00096563	0.03053598	2.74
	9	1.71990361	0.12196826	± 0.00094701	0.02994712	7.09
	10	1.79227413	0.19433878	± 0.00093591	0.02959616	10.84

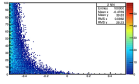
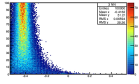
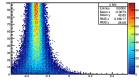
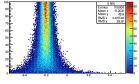
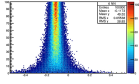
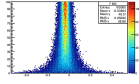
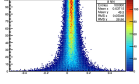
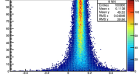
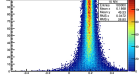
SHREC'13 object 146

	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	1.17765633	-0.53707191	± 0.00200317	0.02627137	-45.61
	3	1.26456878	-0.45015945	± 0.00154017	0.04853367	-35.60
	4	1.38826967	-0.32645857	± 0.00133771	0.04230206	-23.52
	5	1.50365237	-0.21107586	± 0.00124401	0.03933891	-14.04
	6	1.59723832	-0.11748991	± 0.00117551	0.03717277	-7.36
	7	1.68091940	-0.03380884	± 0.00112022	0.03542443	-2.01
	8	1.75864098	0.04391275	± 0.00106408	0.03364913	2.50
	9	1.83903435	0.12430612	± 0.00104420	0.03302056	6.76
	10	1.91401893	0.19929070	± 0.00102743	0.03249025	10.41

SHREC'13 object 205

	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	1.10146738	-0.50880970	± 0.00317991	0.02772183	-46.19
	3	1.17698369	-0.43329339	± 0.00145466	0.04553801	-36.81
	4	1.30149223	-0.30878485	± 0.00126841	0.04011067	-23.73
	5	1.40873444	-0.20154264	± 0.00110314	0.03488444	-14.31
	6	1.49864524	-0.11163184	± 0.00100471	0.03177172	-7.45
	7	1.58350502	-0.02677207	± 0.00094614	0.02991970	-1.69
	8	1.66422949	0.05395240	± 0.00090713	0.02868590	3.24
	9	1.74287693	0.13259985	± 0.00087674	0.02772484	7.61
	10	1.81695637	0.20667929	± 0.00085632	0.02707935	11.38

SHREC'13 object 210

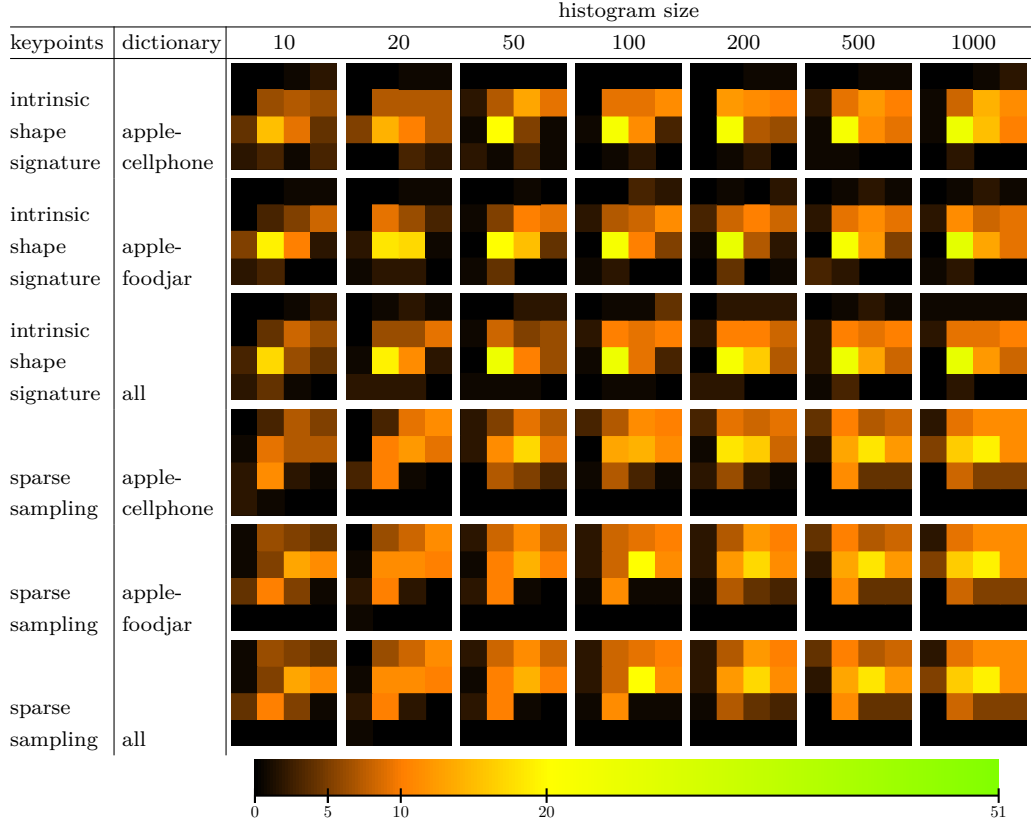
	n.n.	app. mean	mean diff.	mean error	deviation	diff.(%)
	2	1.12197939	-0.51011305	± 0.00190770	0.02751331	-45.47
	3	1.20519327	-0.42689918	± 0.00155335	0.04872675	-35.42
	4	1.31960687	-0.31248558	± 0.00133274	0.04214492	-23.68
	5	1.42565696	-0.20643549	± 0.00119045	0.03764535	-14.48
	6	1.51214232	-0.11995013	± 0.00110590	0.03497155	-7.93
	7	1.59129154	-0.04080090	± 0.00104614	0.03308190	-2.56
	8	1.66709094	0.03499849	± 0.00099471	0.03145546	2.10
	9	1.74407467	0.11198222	± 0.00096867	0.03063199	6.42
	10	1.81726071	0.18516826	± 0.00095025	0.03004961	10.19

C.2.1 3-D Shape Context

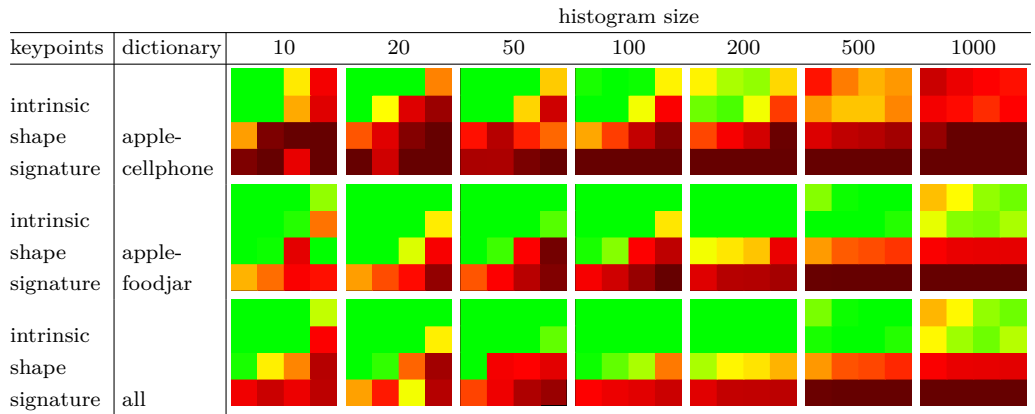
keypoints	dictionary	histogram size						
		10	20	50	100	200	500	1000
intrinsic shape signature	apple- cellphone							
intrinsic shape signature	apple- foodjar							
intrinsic shape signature	all							
sparse sampling	apple- cellphone							
sparse sampling	apple- foodjar							
sparse sampling	all							

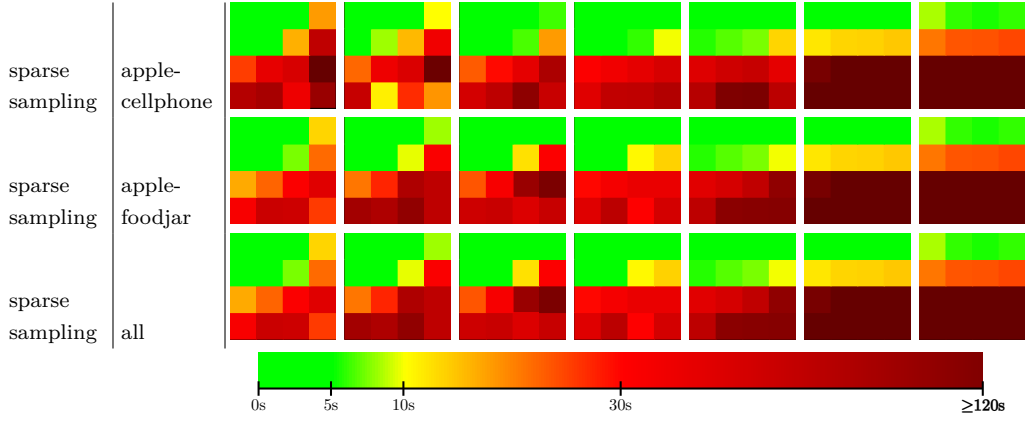
$\leq 80\%$
90%
95%
100%

Number of of object classes where the combination of C and γ leads to the best classification rate of binary classifier support vector machines:

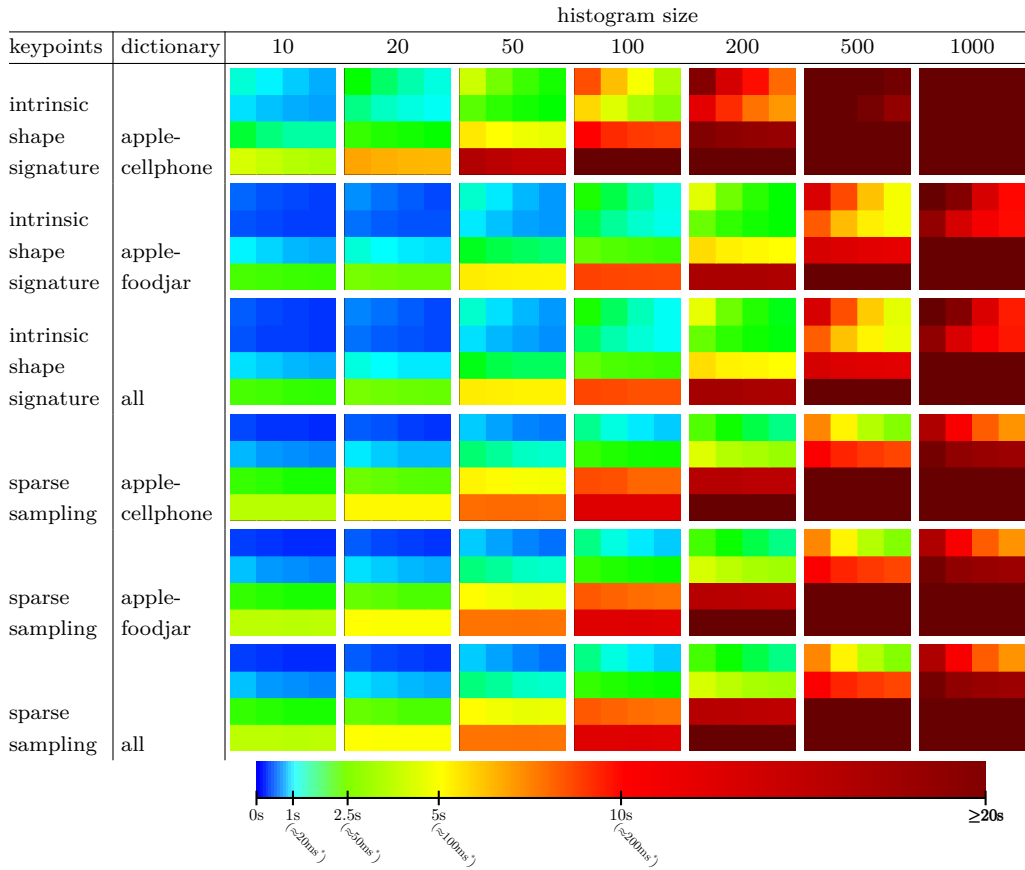


Training times of the support vector machines using different combinations of C and γ :

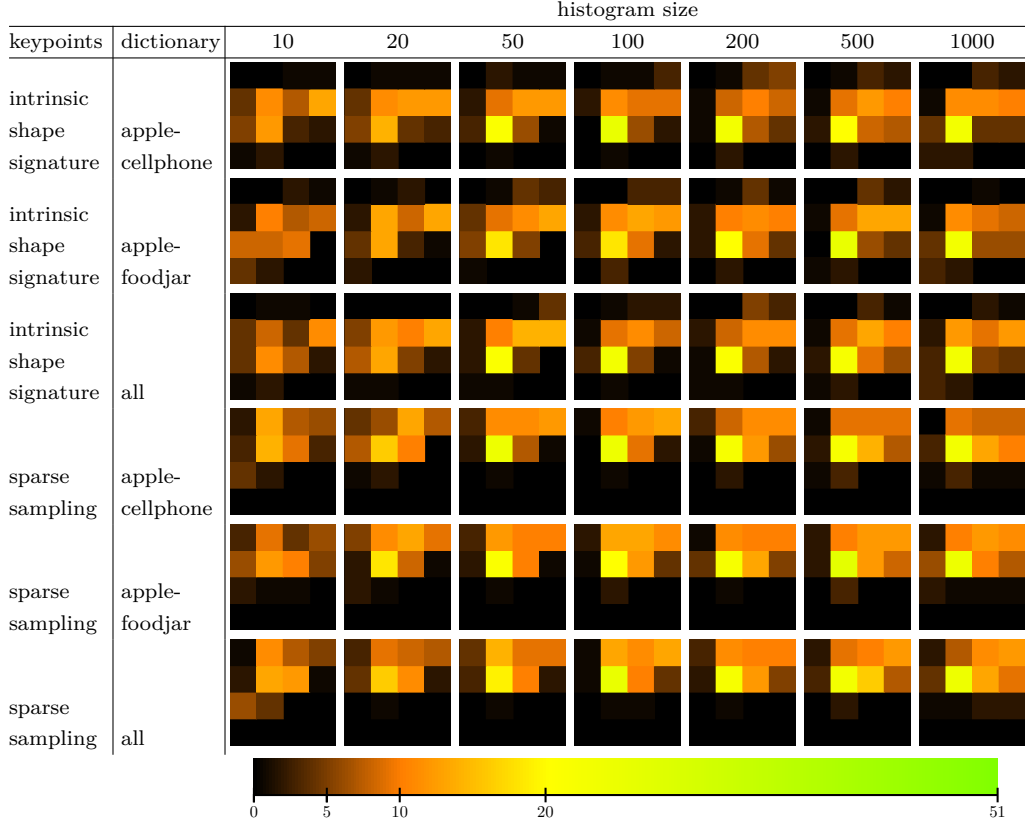




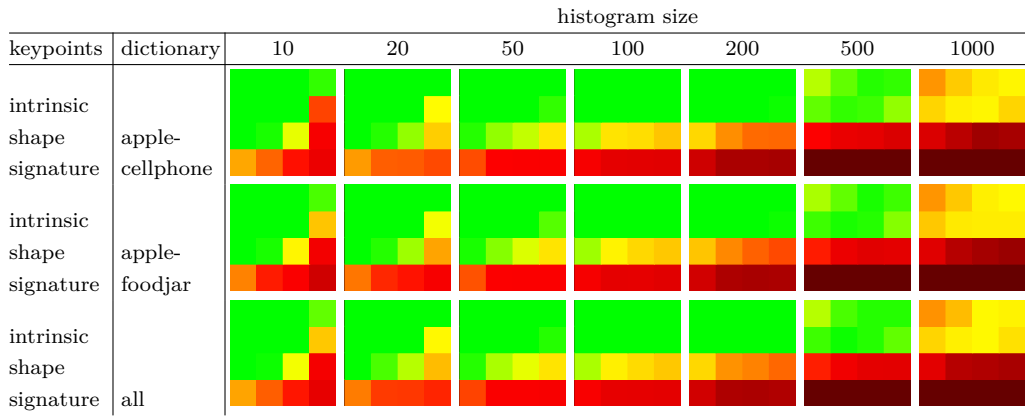
Prediction times of the support vector machines using different combinations of C and γ :

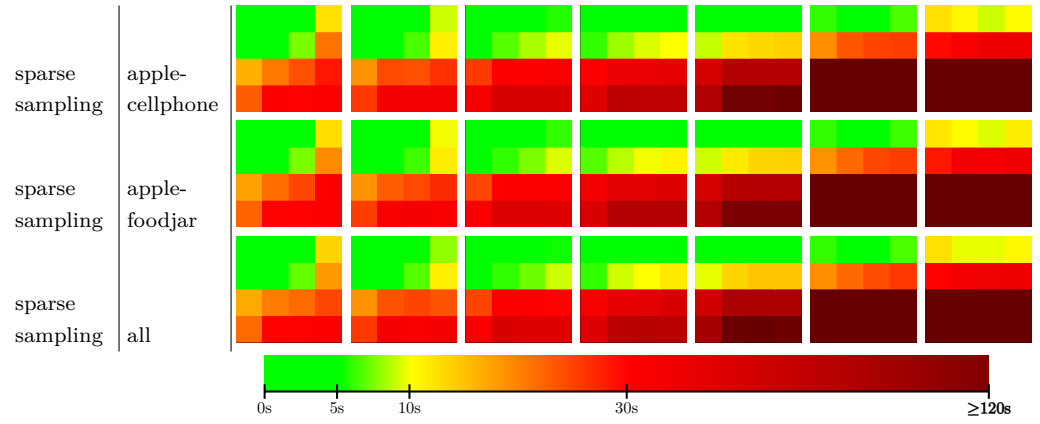


Number of of object classes where the combination of C and γ leads to the best classification rate of binary classifier support vector machines:

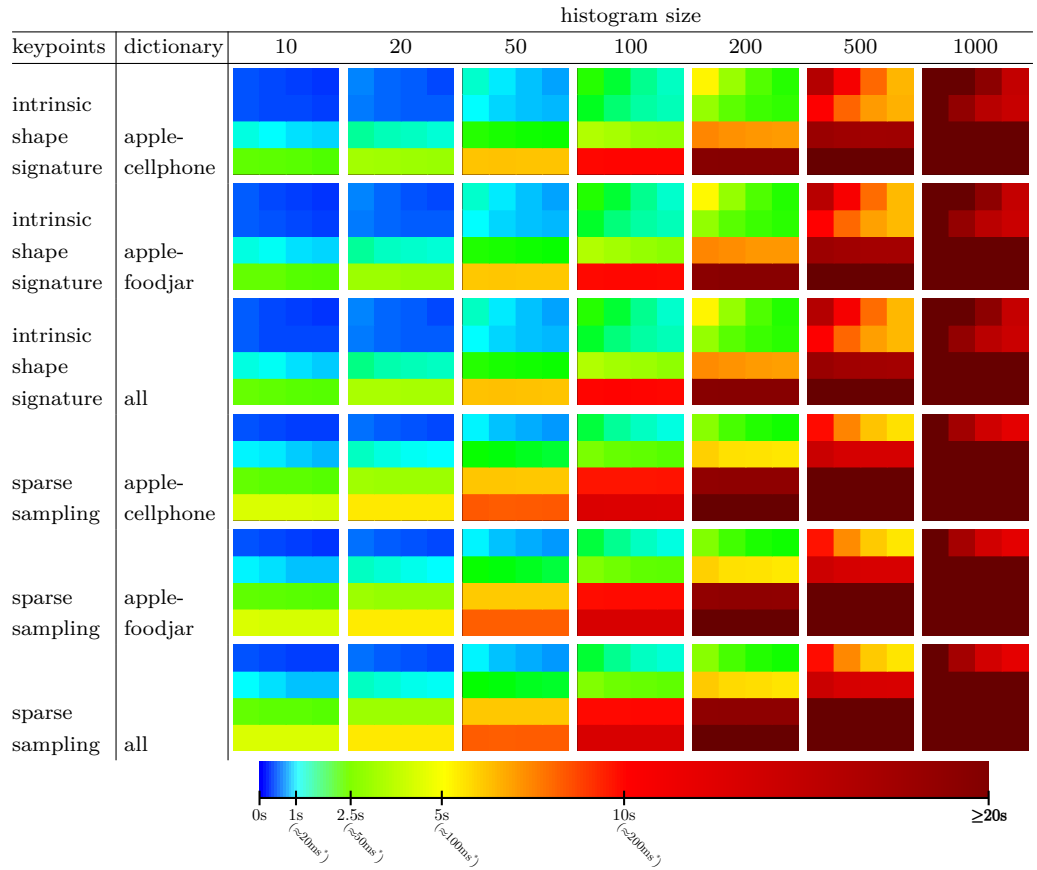


Training times of the support vector machines using different combinations of C and γ :

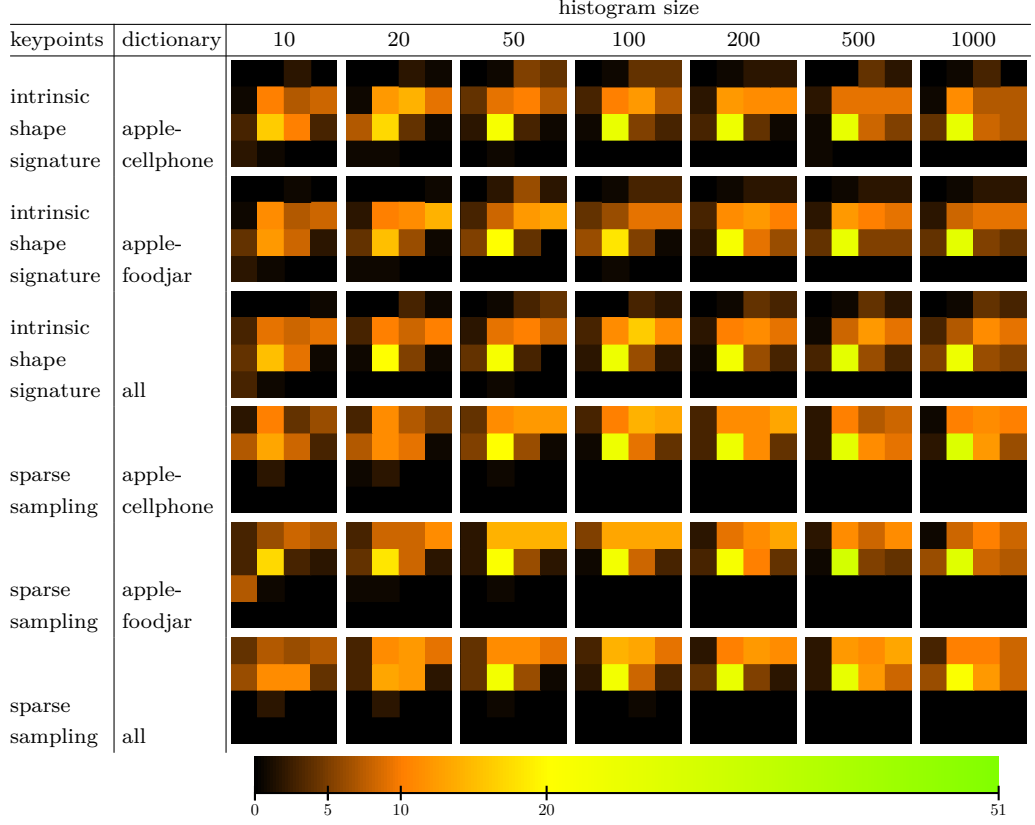




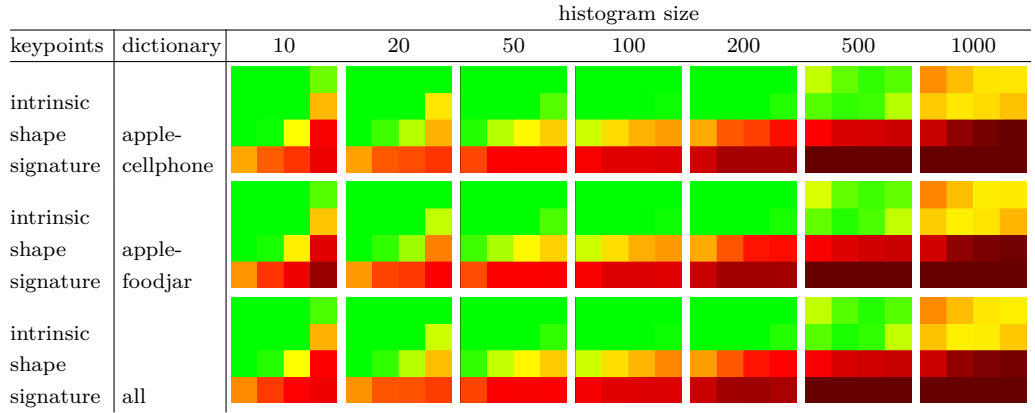
Prediction times of the support vector machines using different combinations of C and γ :

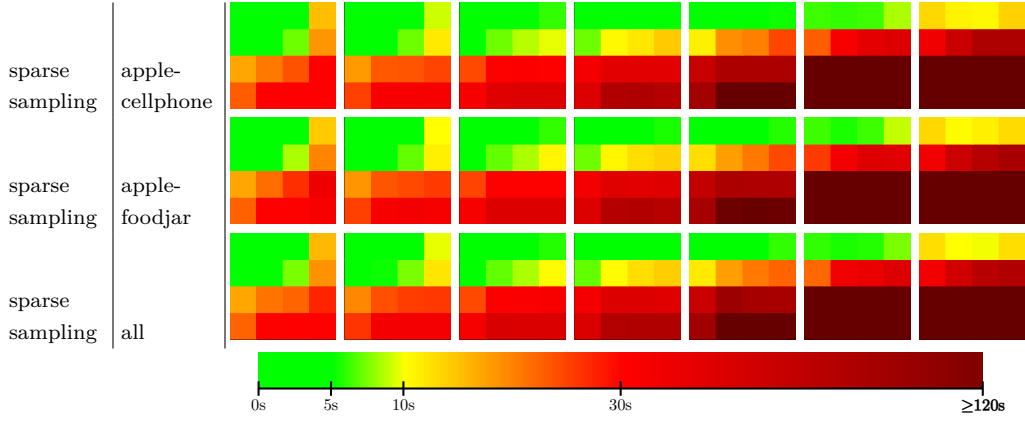


Number of of object classes where the combination of C and γ leads to the best classification rate of binary classifier support vector machines:

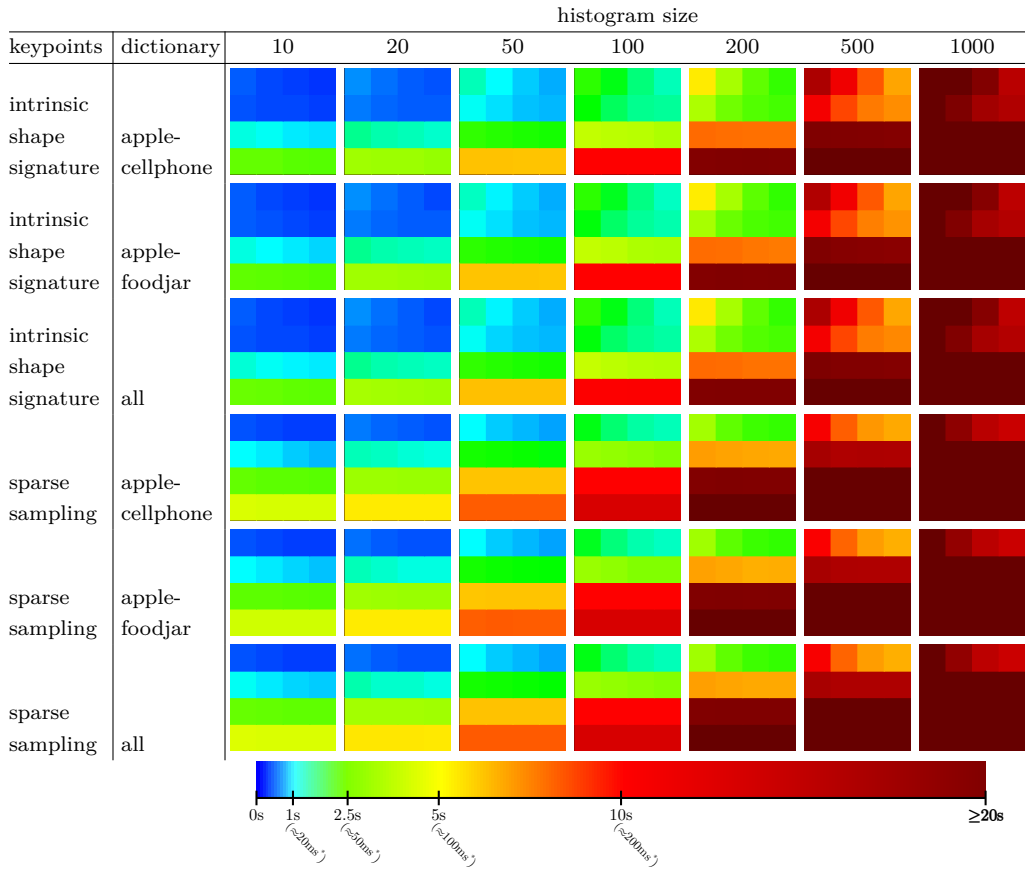


Training times of the support vector machines using different combinations of C and γ :

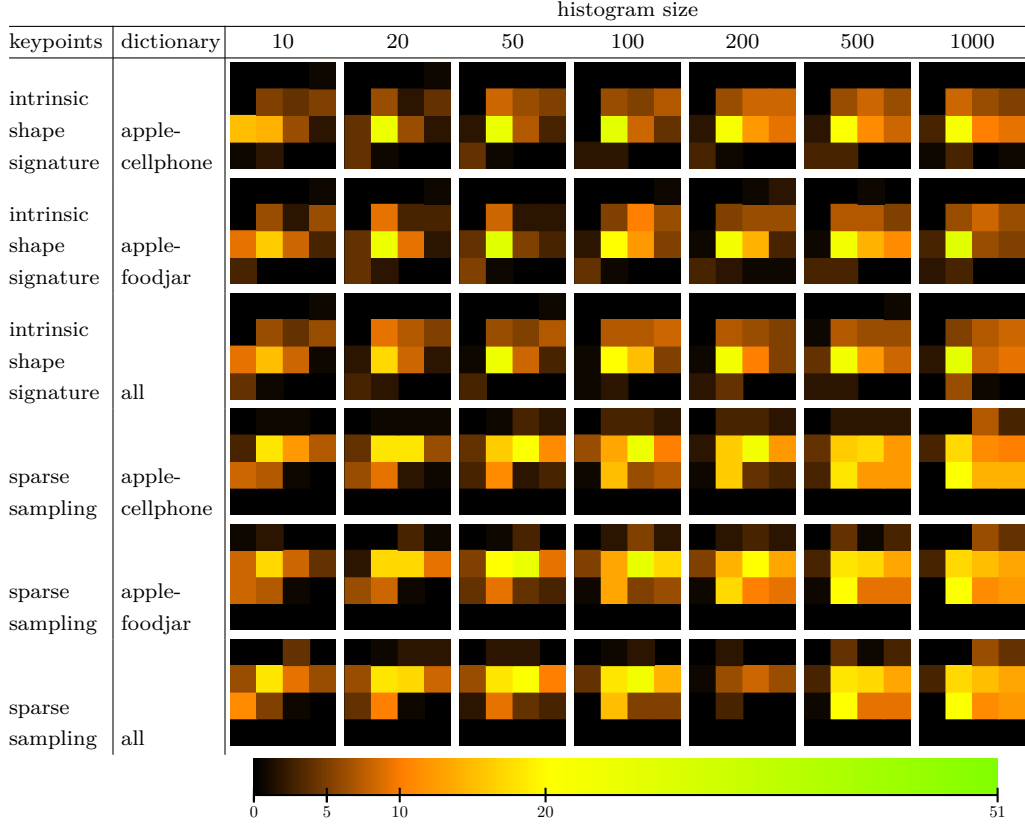




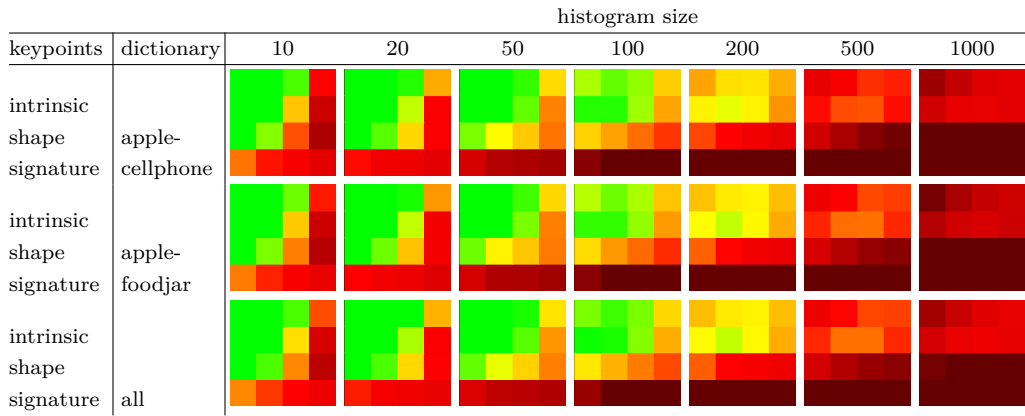
Prediction times of the support vector machines using different combinations of C and γ :

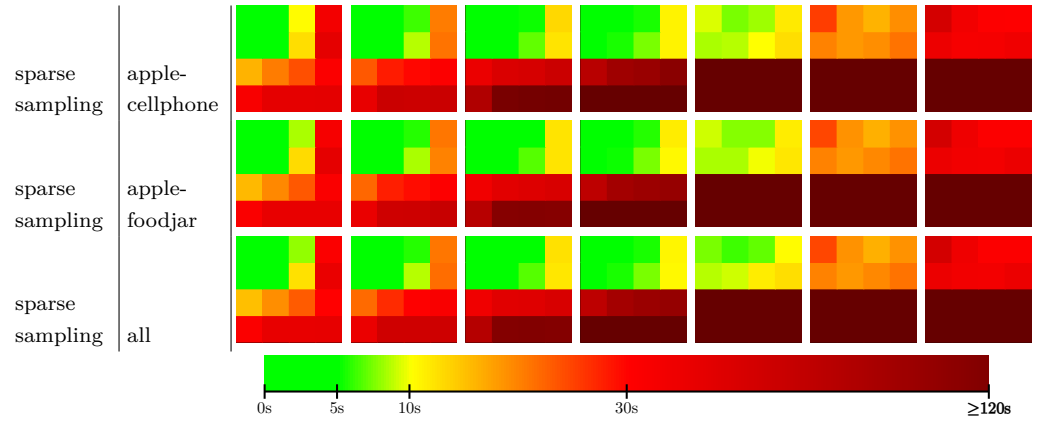


Number of of object classes where the combination of C and γ leads to the best classification rate of binary classifier support vector machines:

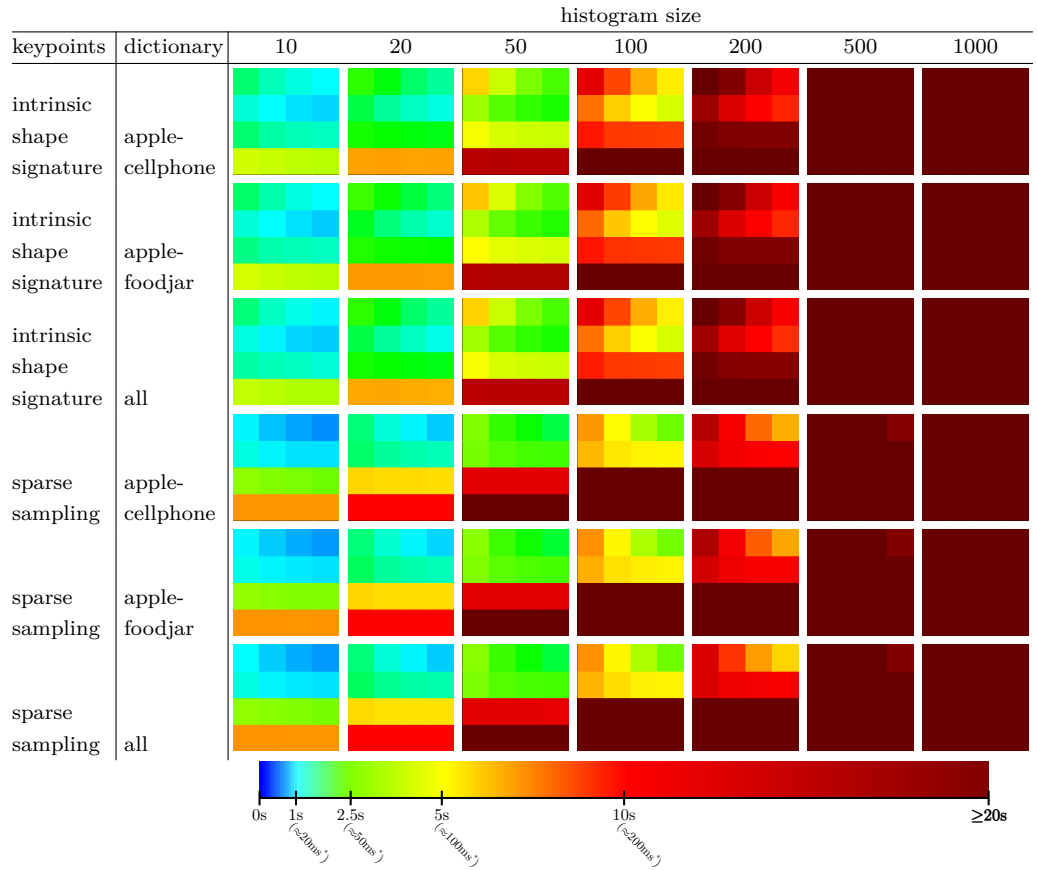


Training times of the support vector machines using different combinations of C and γ :

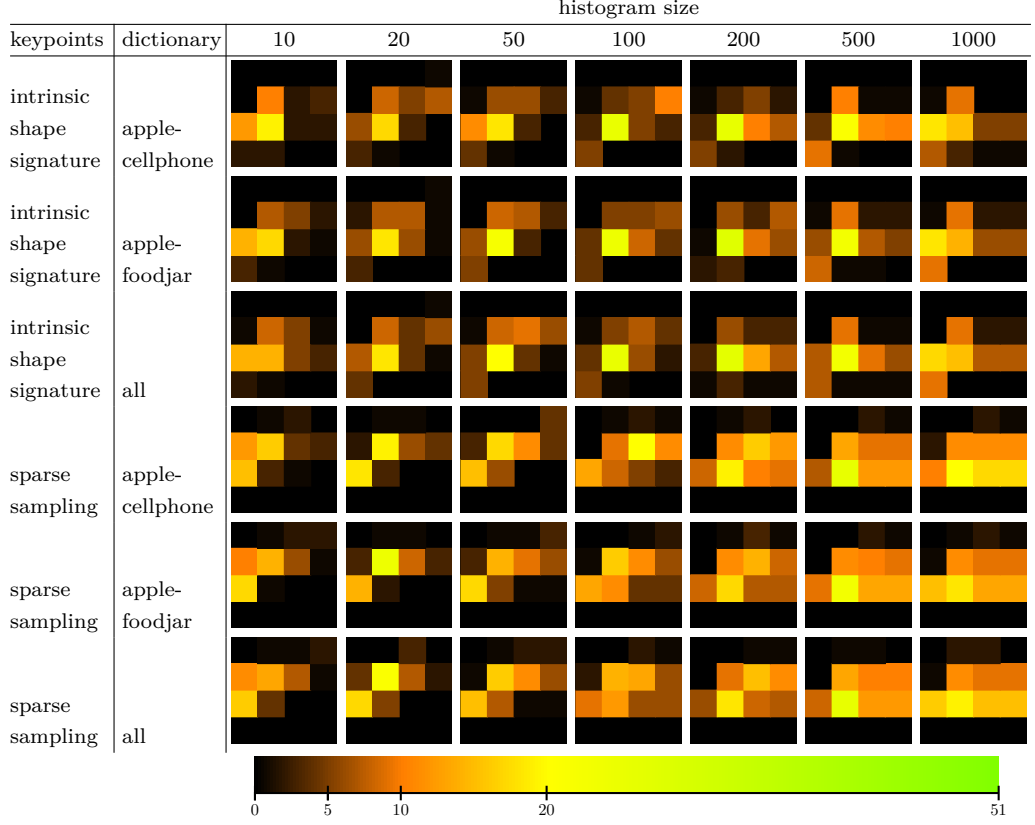




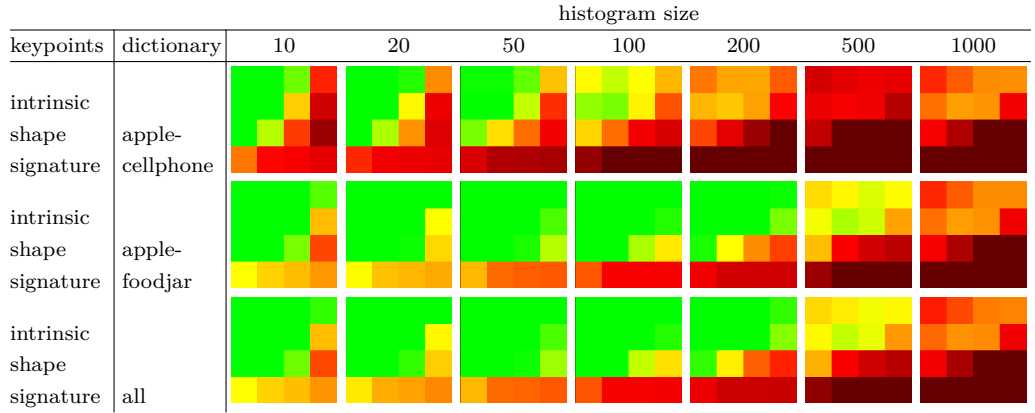
Prediction times of the support vector machines using different combinations of C and γ :

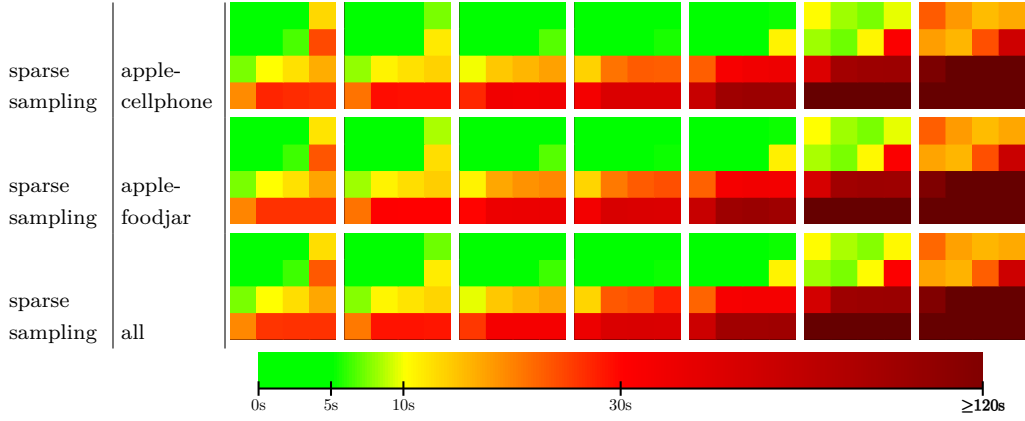


Number of of object classes where the combination of C and γ leads to the best classification rate of binary classifier support vector machines:

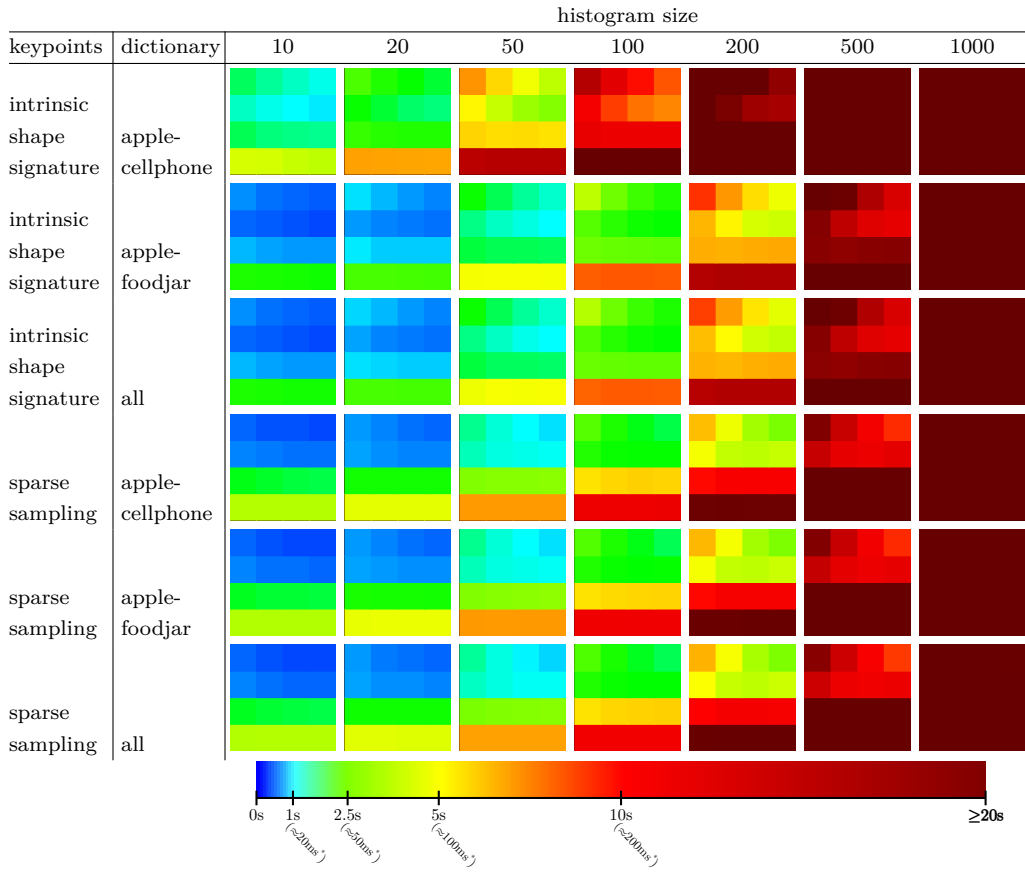


Training times of the support vector machines using different combinations of C and γ :



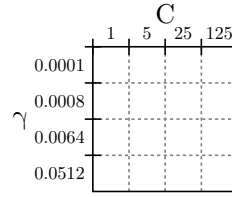


Prediction times of the support vector machines using different combinations of C and γ :

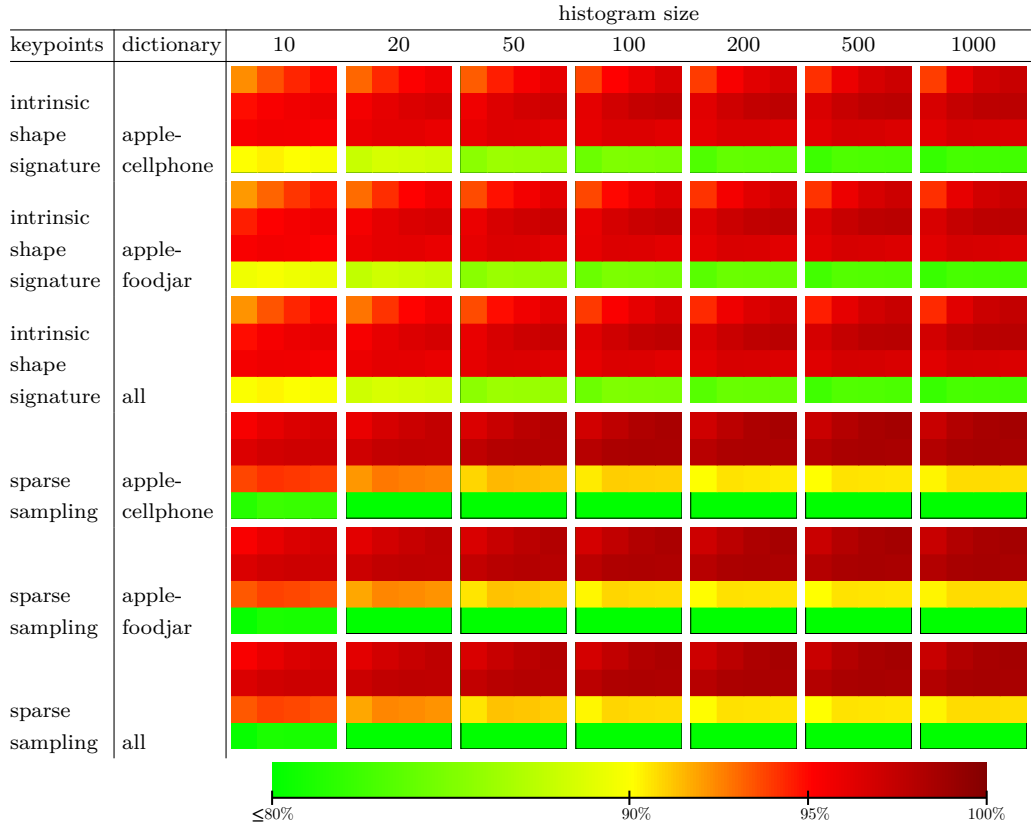


C.2.6 Unique Shape Context

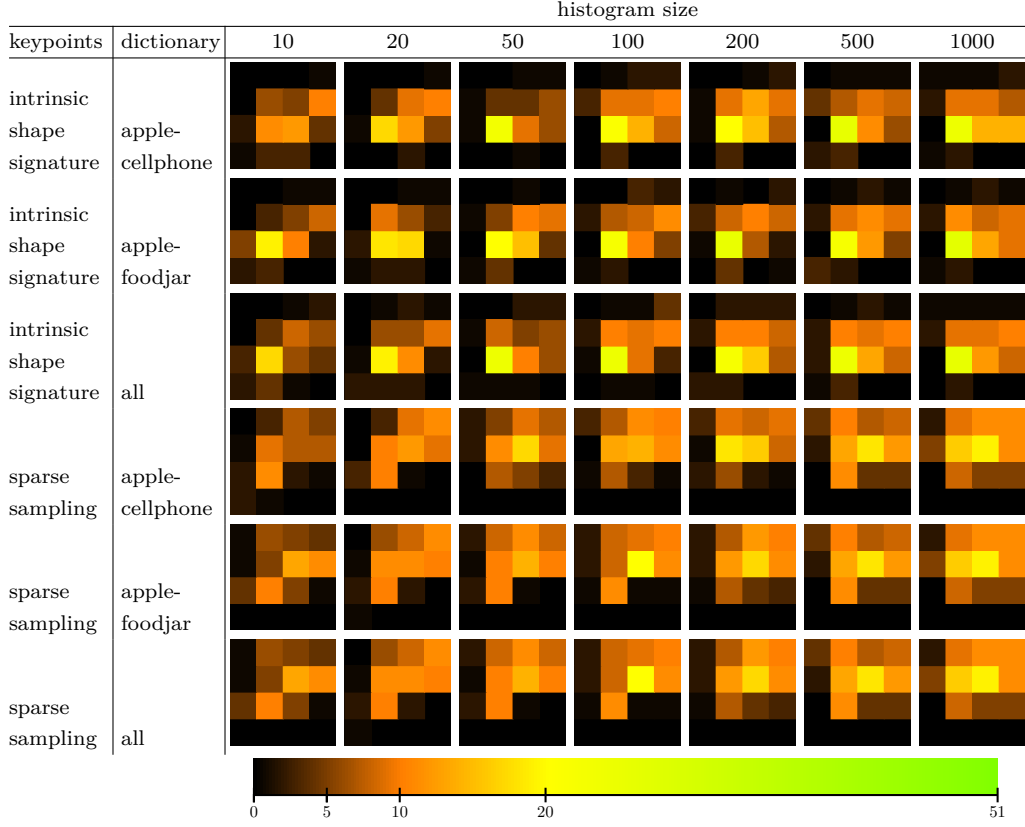
The labels and axes of all histograms concerning the parameter optimization of the basic classification pipeline are shown below. All these histograms have the same axes and labels. To save space the axes and labels are depicted separately and only once. The values of C increase from left to right, while the values of γ increase from top to bottom.



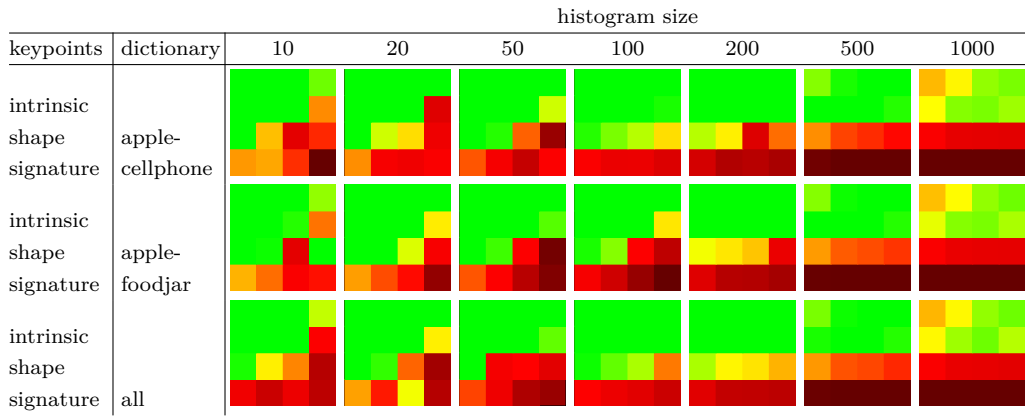
Mean classification rate of binary classifier support vector machines:

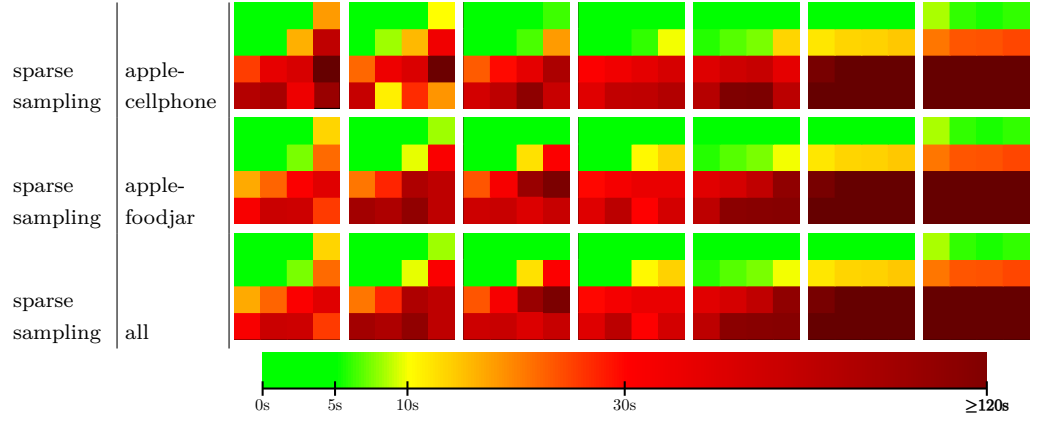


Number of of object classes where the combination of C and γ leads to the best classification rate of binary classifier support vector machines:

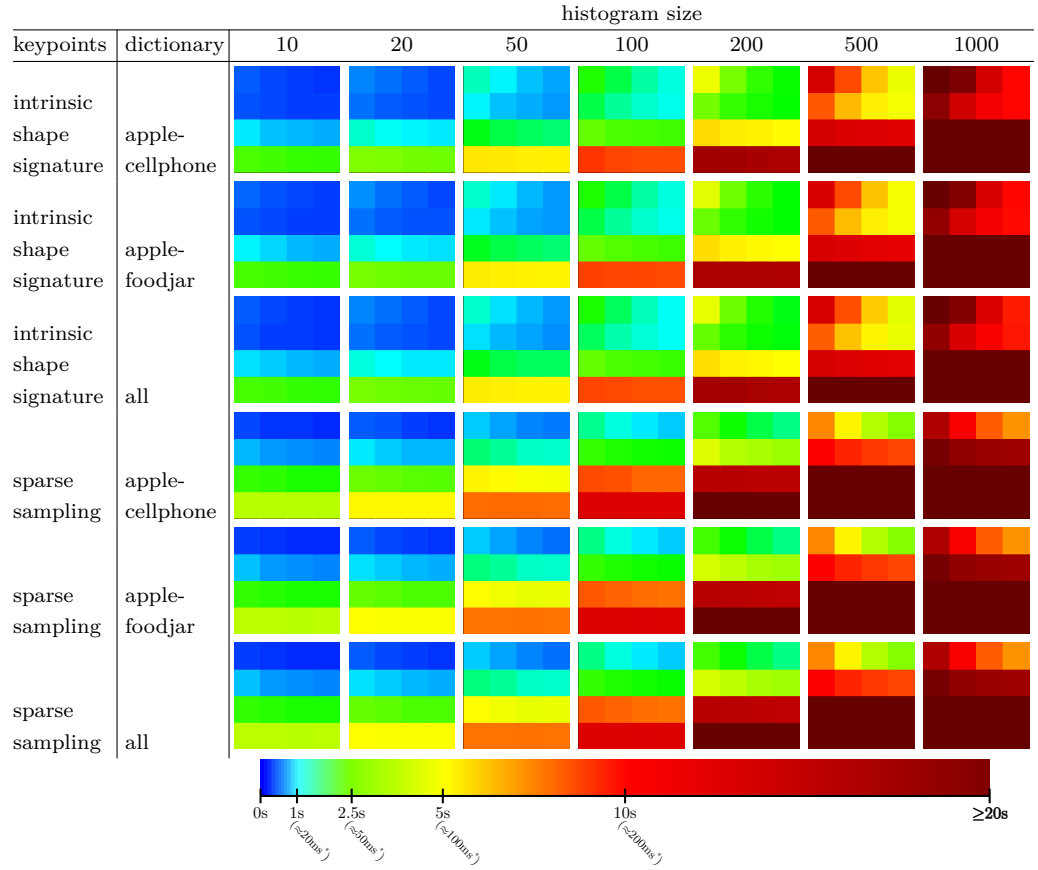


Training times of the support vector machines using different combinations of C and γ :





Prediction times of the support vector machines using different combinations of C and γ :



List of Figures

2.1	3-D classification pipeline	7
2.2	Generic recognition pipeline	9
2.3	Classification pipeline	9
2.4	Viewpoint Component of VFH	14
2.5	Viewpoint Feature Histogram	15
3.1	3-D integral descriptor	19
3.2	Values of normalized $V_r(\mathbf{p})$	20
3.3	Extended Integral Volume Descriptors	22
3.4	Local surface patches	24
3.5	NARF borders	30
3.6	NARF border weight	30
3.7	NARF keypoint	31
4.1	Calculation of spin images	36
4.2	Example of spin images	37
4.3	3-D shape context	38
4.4	Local surface patches	39
4.5	Thrift	40
4.6	PFH	41
4.7	FPFH	43
4.8	2.5-D SIFT	45

4.9	ISS	46
4.10	NARF Descriptor	47
4.11	Spherical signature of SHOT descriptor	50
4.12	SURE	51
4.13	HONV descriptor	53
5.1	SVM example 1	56
5.2	SVM example 2	56
5.3	SVM example 3	57
6.1	Reinforcement learning	61
6.2	Markov Decision Process	62
6.3	Exploration-Exploitation Example 1	67
6.4	Exploration-Exploitation Dilemma	68
6.5	Upper Confidence Bound	69
7.1	Baseline experiment pipeline	72
7.2	Construction of the of the reinforcement learning model - part 1	78
7.3	Construction of the of the reinforcement learning model - part 2	79
7.4	Construction of the of the reinforcement learning model - part 3	81
7.5	Construction of the of the reinforcement learning model - part 4	83
7.6	RGB-D Object Dataset	85
7.7	RGB-D Object Dataset v.2	85
7.8	Distribution of Point Cloud Sizes	86
7.9	Distribution of Point Cloud Resolutions	87
8.1	Baseline experiment pipeline	91
8.2	Point cloud resolution – different number of nearest neighbors	95
8.3	Distribution of keypoints	100
8.4	Computation times for a single keypoint	100

8.5	Computation time for keypoint of an object	101
8.6	Relation between computation time and keypoints per object . . .	102
8.7	Different spin image placements	103
8.8	Computation times for local 3-D feature descriptors	109
8.9	Computation times for bag of features vocabularies	111
8.10	Axes and labels of all SVM training histograms	113
8.11	Mean classification results for different SVM parameters	114
8.12	Classification results for different SVM parameters	115
8.13	Training times for different SVM parameters	116
8.14	Prediction times for different SVM parameters	116
8.15	Mean classification results – comparison of object classes used for vocabularies	118
8.16	Mean classification results – comparison of keypoints	119
8.17	Mean classification rates for different local 3-D feature descriptors	122
8.18	Prediction times for different local 3-D feature descriptors	123
8.19	Precision-recall graphs of all classifiers	125
8.20	Precision-recall graphs with different limits	127
9.1	Fusion with Reinforcement Learning	134
9.2	The number of states and Q -values during RL training – different limits	136
9.3	Influence of the learned Q -values on the classification results – different limits	137
9.4	The number of states and Q -values during RL training – different limits	139
9.5	Classification results using only 10 object classes – different limits .	140
9.6	The number of states and Q -values during RL training – unique limits	141
9.7	Influence of the learned Q -values on the classification results – unique limits	142

9.8	The number of states and Q -values during RL training – unique limits	143
9.9	Classification results using only 10 object classes – unique limits .	144
9.10	Differentiation of the First State	146
9.11	Classification results with global properties – all classes	146
9.12	Classification results with global properties – 10 classes	147
9.13	Comparison of adaptive learning for different algorithms	151
9.14	Classification results without SHOT and SI	152

Bibliography

- [1] Anders Adamson and Marc Alexa. Ray tracing point set surfaces. In *Shape Modeling International*, pages 272–279. IEEE, 2003.
- [2] A. Aldoma, Zoltan-Csaba Marton, F. Tombari, W. Wohlkinger, C. Potthast, B. Zeisl, R.B. Rusu, S. Gedikli, and M. Vincze. Tutorial: Point cloud library: Three-dimensional object recognition and 6 DOF pose estimation. *Robotics Automation Magazine, IEEE*, 19(3):80–91, Sept 2012.
- [3] Aitor Aldoma, Federico Tombari, Radu Bogdan Rusu, and Markus Vincze. OUR-CVFH – oriented, unique and repeatable clustered viewpoint feature histogram for object recognition and 6 DOF pose estimation. In *Pattern Recognition*, pages 113–122. Springer, 2012.
- [4] Luis A Alexandre. 3D descriptors for object and category recognition: a comparative evaluation. In *Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vilamoura, Portugal*, 2012.
- [5] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.
- [6] Neslihan Bayramoglu and A Aydin Alatan. Shape index SIFT: Range image recognition using local features. In *Proceedings of the 20th International Conference on Pattern Recognition (ICPR)*, pages 352–355. IEEE, 2010.
- [7] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
- [8] Alexander M. Bronstein, Michael M. Bronstein, and Maks Ovsjanikov. *3D Imaging, Analysis and Applications*, chapter Feature-Based Methods in 3D Shape Analysis, pages 185–219. Springer London, London, 2012.

- [9] Benjamin Bustos, Daniel Keim, Dietmar Saupe, Tobias Schreck, and Dejan Vranić. An experimental effectiveness comparison of methods for 3D similarity search. *International Journal on Digital Libraries*, 6(1):39–54, 2006.
- [10] Hui Chen and Bir Bhanu. 3D free-form object recognition in range images using local surface patches. *Pattern Recognition Letters*, 28(10):1252–1262, 2007.
- [11] Flavio Chierichetti, Ravi Kumar, Sandeep Pandey, and Sergei Vassilvitskii. Finding the jaccard median. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 293–311. Society for Industrial and Applied Mathematics, 2010.
- [12] Michal Cholewa and Przemyslaw Sporysz. Classification of dynamic sequences of 3D point clouds. In *Artificial Intelligence and Soft Computing*, pages 672–683. Springer, 2014.
- [13] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312. ACM, 1996.
- [14] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893. IEEE, 2005.
- [15] John D’Errico. Surface fitting using gridfit. MATLAB Central File Exchange, 2008.
- [16] Chitra Dorai and Anil K Jain. Cosmos-a representation scheme for free-form surfaces. In *Proceedings of the Fifth International Conference on Computer Vision*, pages 1024–1029. IEEE, 1995.
- [17] Bertram Drost, Markus Ulrich, Nassir Navab, and Slobodan Ilic. Model globally, match locally: Efficient and robust 3D object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 998–1005. IEEE, 2010.

- [18] Helin Dutagaci, Chun Pan Cheung, and Afzal Godil. Evaluation of 3D interest point detection techniques via human-generated ground truth. *The Visual Computer*, 28(9):901–917, 2012.
- [19] Silvio Filipe and Luís A. Alexandre. A comparative evaluation of 3D key-point detectors. In *Proceedings of the 9th Conference on Telecommunications, Conftele*, 2013.
- [20] Torsten Fiolka, Jörg Stücker, Dominik A Klein, Dirk Schulz, and Sven Behnke. Sure: Surface entropy for distinctive 3D features. In *Spatial Cognition VIII*, pages 74–93. Springer, 2012.
- [21] Torsten Fiolka, Jorg Stuckler, Dominik A Klein, Dirk Schulz, and Sven Behnke. Distinctive 3D surface entropy features for place recognition. In *Proceedings of the European Conference on Mobile Robots (ECMR)*, pages 204–209. IEEE, 2013.
- [22] Alex Flint, Anthony Dick, and Anton van den Hengel. Thrift: Local 3D structure recognition. In *Proceedings of the 9th Biennial Conference of the Australian Pattern Recognition Society on Digital Image Computing Techniques and Applications*, pages 182–188. IEEE, 2007.
- [23] Andrea Frome, Daniel Huber, Ravi Kolluri, Thomas Bulow, and Jitendra Malik. Recognizing objects in range data using regional point descriptors. In *Proceedings of the European Conference on Computer Vision (ECCV)*, May 2004.
- [24] Jens Garstka and Gabriele Peters. Learning Strategies to Select Point Cloud Descriptors for 3D Object Classification: A Proposal. In *Eurographics 2014 - Posters*. The Eurographics Association, 2014.
- [25] Jens Garstka and Gabriele Peters. Fast and robust keypoint detection in unstructured 3-D point clouds. In *Proceedings of the 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, July 2015.
- [26] Natasha Gelfand, Niloy J Mitra, Leonidas J Guibas, and Helmut Pottmann. Robust global registration. In *Symposium on geometry processing*, volume 2, page 5, 2005.

- [27] Yulan Guo, Mohammed Bennamoun, Ferdous 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*, 99(PrePrints):1, 2014.
- [28] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Proceedings of the Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.
- [29] M. Heczko, D. Keim, D. Saupe, and D. Vranic. Methods for similarity search on 3D databases. *Datenbank-Spektrum (in German)*, 2(2):54–63, 2002.
- [30] Paul Heider, Alain Pierre-Pierre, Ruosi Li, and Cindy Grimm. Local shape descriptors, a survey and evaluation. In *Proceedings of the 4th Eurographics conference on 3D Object Retrieval*, pages 49–56. Eurographics Association, 2011.
- [31] Guenter Hetzl, Bastian Leibe, Paul Levi, and Bernt Schiele. 3D object recognition from range images using local feature histograms. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 394–399, 2001.
- [32] Michael Himmelsbach, Thorsten Luetzel, and Hans-Joachim Wuensche. Real-time object classification in 3D point clouds using point feature histograms. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 994–1000, St. Louis, MO, USA, October 2009. IEEE.
- [33] Alexander Hornung and Leif Kobbelt. Robust reconstruction of watertight 3D models from non-uniformly sampled point clouds without normal information. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, pages 41–50, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.
- [34] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University, July 2003.
- [35] Allison Janoch, Sergey Karayev, Yangqing Jia, Jonathan T Barron, Mario Fritz, Kate Saenko, and Trevor Darrell. A category-level 3D object dataset:

- Putting the kinect to work. In *Consumer Depth Cameras for Computer Vision*, pages 141–165. Springer, 2013.
- [36] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Improving bag-of-features for large scale image search. *International Journal of Computer Vision*, 87(3):316–336, 2010.
- [37] A.E. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(5):433–449, 1999.
- [38] Andrew Edie Johnson and Martial Hebert. Surface matching for object recognition in complex three-dimensional scenes. *Image and Vision Computing*, 16(9):635–651, 1998.
- [39] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. *arXiv preprint arXiv:1412.2306*, 2014.
- [40] Daniel A Keim. *Efficient geometry-based similarity search of 3D spatial databases*, volume 28. ACM, 1999.
- [41] Jan Knopp, Mukta Prasad, Geert Willems, Radu Timofte, and Luc Van Gool. Hough transform and 3D SURF for robust three dimensional classification. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 589–602. Springer, 2010.
- [42] Vijay R Konda and John N Tsitsiklis. On actor-critic algorithms. *SIAM journal on Control and Optimization*, 42(4):1143–1166, 2003.
- [43] Shu Kong and Donghui Wang. A dictionary learning approach for classification: Separating the particularity and the commonality. In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Proceedings of the European Conference on Computer Vision (ECCV)*, volume 7572 of *Lecture Notes in Computer Science*, pages 186–199. Springer Berlin Heidelberg, 2012.
- [44] Kevin Lai, Liefeng Bo, and Dieter Fox. Unsupervised feature learning for 3D scene labeling. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2014.
- [45] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. A large-scale hierarchical multi-view RGB-D object dataset. In *Proceedings of the IEEE*

- International Conference on Robotics and Automation (ICRA)*, pages 1817–1824. IEEE, 2011.
- [46] Tony Lindeberg. Feature detection with automatic scale selection. *International journal of computer vision*, 30(2):79–116, 1998.
- [47] Stuart Lloyd. Least squares quantization in PCM. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.
- [48] Tsz-Wai Rachel Lo and J. Paul Siebert. Local feature extraction and matching on range images: 2.5D SIFT. *Computer Vision and Image Understanding*, 113(12):1235 – 1250, 2009. Special issue on 3D Representation for Object and Scene Recognition.
- [49] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.
- [50] Luca Lucchese, Gianfranco Doretto, and Guido M. Cortelazzo. A frequency domain technique for range data registration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(11):1468–1484, 2002.
- [51] Marianna Madry, Heydar Maboudi Afkham, Carl Henrik Ek, Stefan Carlsson, and Danica Kragic. Extracting essential local object characteristics for 3D object categorization. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2240–2247. IEEE, 2013.
- [52] Marianna Madry, Carl Henrik Ek, Renaud Detry, Kaiyu Hang, and Danica Kragic. Improving generalization for 3D object categorization with global structure histograms. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1379–1386. IEEE, 2012.
- [53] Siddharth Manay, Byung-Woo Hong, Anthony J Yezzi, and Stefano Soatto. *Integral invariant signatures*. Springer, 2004.
- [54] B. Matei, Ying Shan, H.S. Sawhney, Yi Tan, R. Kumar, Daniel 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*, 28(7):1111 – 1126, July 2006.

- [55] James Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character*, pages 415–446, 1909.
- [56] A. Mian, M. Bennamoun, and R. Owens. On the repeatability and quality of keypoints for local feature-based 3D object retrieval from cluttered scenes. *International Journal of Computer Vision*, 89(2-3):348–361, 2010.
- [57] Ajmal S Mian, Mohammed Bennamoun, and Robyn Owens. Three-dimensional model-based object recognition and segmentation in cluttered scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1584–1601, 2006.
- [58] Eric Paquet and Marc Rioux. Nefertiti: a query by content system for three-dimensional model and image databases management. *Image and Vision Computing*, 17(2):157–166, 1999.
- [59] Eric Paquet, Marc Rioux, Anil Murching, Thumpudi Naveen, and Ali Tabatabai. Description of shape information for 2-D and 3-D objects. *Signal Processing: Image Communication*, 16(1):103–122, 2000.
- [60] Mark Pauly, Richard Keiser, and Markus Gross. Multi-scale feature extraction on point-sampled surfaces. In *Computer graphics forum*, volume 22, pages 281–289. Wiley Online Library, 2003.
- [61] Tahir Rabbani Shah. *Automatic reconstruction of industrial installations: Using point clouds and images*. PhD thesis, TU Delft, 2006.
- [62] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979.
- [63] Emanuele Rodolà, Andrea Albarelli, Filippo Bergamasco, and Andrea Torsello. A scale independent selection process for 3D object recognition in cluttered scenes. *International Journal of Computer Vision*, 102(1-3):129–145, 2013.
- [64] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Always learning. Pearson, 2013.

- [65] Radu Bogdan Rusu. Semantic 3D object maps for everyday manipulation in human living environments. *KI-Künstliche Intelligenz*, 24(4):345–348, 2010.
- [66] Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. Aligning point cloud views using persistent feature histograms. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3384–3391. IEEE, 2008.
- [67] Radu Bogdan Rusu, Gary Bradski, Romain Thibaux, and John Hsu. Fast 3D recognition and pose using the viewpoint feature histogram. In *Proceedings of the 23rd IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2155–2162. IEEE, 2010.
- [68] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (PCL). In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–4. IEEE, 2011.
- [69] Radu Bogdan Rusu, Andreas Holzbach, Michael Beetz, and Gary Bradski. Detecting and segmenting objects for mobile manipulation. In *Proceedings of the IEEE 12th International Conference on Computer Vision Workshops (ICCV)*, pages 47–54. IEEE, 2009.
- [70] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Learning informative point classes for the acquisition of object model maps. In *Proceedings of the 10th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 643–650. IEEE, 2008.
- [71] R.B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (FPFH) for 3D registration. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3212–3217, 2009.
- [72] Samuele Salti, Federico Tombari, and Luigi Di Stefano. A performance evaluation of 3D keypoint detectors. In *Proceedings of the International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)*, pages 236–243. IEEE, 2011.
- [73] Samuele Salti, Federico Tombari, and Luigi Di Stefano. SHOT: Unique signatures of histograms for surface and texture description. *Computer Vision and Image Understanding*, 125(0):251–264, 2014.

- [74] Dietmar Saupe and Dejan V Vranić. 3D model retrieval with spherical harmonics and moments. In *Pattern Recognition*, pages 392–397. Springer, 2001.
- [75] Viktor Seib, Susanne Christ-Friedmann, Susanne Thierfelder, and Dietrich Paulus. Object class and instance recognition on RGB-D data. In *Proceedings of the Sixth International Conference on Machine Vision (ICMV)*. International Society for Optics and Photonics, 2013.
- [76] SHREC’13 - 3D shape retrieval contest 2013. <http://3dorus.ist.utl.pt/research/BeKi/index.html>.
- [77] Stephen M Smith and J Michael Brady. SUSAN – a new approach to low level image processing. *International journal of computer vision*, 23(1):45–78, 1997.
- [78] The stanford 3D scanning repository. <http://graphics.stanford.edu/data/3Dscanrep/>.
- [79] Bastian Steder, Radu Bogdan Rusu, Kurt Konolige, and Wolfram Burgard. NARF: 3D range image features for object recognition. In *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 44, 2010.
- [80] Bastian Steder, Radu Bogdan Rusu, Kurt Konolige, and Wolfram Burgard. Point feature extraction on 3D range scans taking into account object boundaries. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2601–2608. IEEE, 2011.
- [81] Hugo Steinhaus. *Mathematical snapshots*. Courier Corporation, 2012.
- [82] J Stuckler and Sven Behnke. Interest point detection in depth images through scale-space surface analysis. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3568–3574. IEEE, 2011.
- [83] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998.
- [84] Richard Stuart Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, 1984.

- [85] Motofumi T Suzuki, Toshikazu Kato, and Nobuyuki Otsu. A similarity retrieval of 3D polygonal models using rotation invariant shape descriptors. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, volume 4, pages 2946–2952. IEEE, 2000.
- [86] Chwen-Jye Sze, H-YM Liao, Hai-Lung Hung, Kuo-Chin Fan, and Jun-Wei Hsieh. Multiscale edge detection on range images via normal changes. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 45(8):1087–1092, 1998.
- [87] Csaba Szepesvári. *Algorithms for Reinforcement Learning*, volume 4. Morgan & Claypool Publishers, 2010.
- [88] Babak Taati, Michel Bondy, Piotr Jasiobedzki, and Michael Greenspan. Variable dimensional local shape descriptors for object recognition in range data. In *Proceedings of the IEEE 11th International Conference on Computer Vision (ICCV)*, pages 1–8. IEEE, 2007.
- [89] Shuai Tang, Xiaoyu Wang, Xutao Lv, Tony X Han, James Keller, Zhihai He, Marjorie Skubic, and Shihong Lao. Histogram of oriented normal vectors for object recognition with a depth sensor. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, pages 525–538. Springer, 2013.
- [90] Roberto Toldo, Umberto Castellani, and Andrea Fusiello. A bag of words approach for 3d object categorization. In *Computer Vision/Computer Graphics Collaboration Techniques*, pages 116–127. Springer, 2009.
- [91] Roberto Toldo, Umberto Castellani, and Andrea Fusiello. The bag of words approach for retrieval and categorization of 3D objects. *The Visual Computer*, 26(10):1257–1268, 2010.
- [92] Federico Tombari, Samuele Salti, and Luigi Di Stefano. Unique shape context for 3D data description. In *Proceedings of the ACM workshop on 3D object retrieval*, pages 57–62. ACM, 2010.
- [93] Federico Tombari, Samuele Salti, and Luigi Di Stefano. Unique signatures of histograms for local surface description. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 356–369. Springer, 2010.
- [94] Federico Tombari, Samuele Salti, and Luigi Di Stefano. A combined texture-shape descriptor for enhanced 3d feature matching. In *Proceedings of the*

- 18th IEEE International Conference on Image Processing (ICIP)*, pages 809–812. IEEE, 2011.
- [95] Ranjith Unnikrishnan and Martial Hebert. Multi-scale interest regions from unorganized point clouds. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1–8. IEEE, 2008.
- [96] Vladimir N Vapnik and A Ja Chervonenkis. Theory of pattern recognition. *Nauka*, 1974.
- [97] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. *arXiv preprint arXiv:1411.4555*, 2014.
- [98] Dejan V Vranić and D Saupe. 3D shape descriptor based on 3D fourier transform. In *Proceedings of the EURASIP conference on digital signal processing for multimedia communications and services (ECMCS), Budapest, Hungary*, 2001.
- [99] Dejan V Vranic and Dietmar Saupe. Description of 3D-shape using a complex function on the sphere. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, volume 1, pages 177–180. IEEE, 2002.
- [100] Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.
- [101] Marco Wiering and Martijn Van Otterlo. Reinforcement learning. In *Adaptation, Learning, and Optimization*, volume 12. Springer, 2012.
- [102] Chin-Chia Wu and Sheng-Fuu Lin. Efficient model detection in point cloud data based on bag of words classification. *Journal of Computational Information Systems*, 7(12):4170–4177, 2011.
- [103] Yang Yi, Yan Guang, Zhu Hao, Fu Meng-Yin, and Wang Mei-ling. Object segmentation and recognition in 3D point cloud with language model. In *Proceedings of the International Conference on Multisensor Fusion and Information Integration for Intelligent Systems (MFI)*, pages 1–6. IEEE, 2014.

- [104] Andrei Zaharescu, Edmond Boyer, Kiran Varanasi, and Radu Horaud. Surface feature detection and description with applications to mesh matching. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 373–380. IEEE, 2009.
- [105] Yu Zhong. Intrinsic shape signatures: A shape descriptor for 3D object recognition. In *Proceedings of the IEEE 12th International Conference on Computer Vision Workshops (ICCV)*, pages 689–696. IEEE, 2009.

Lebenslauf

Persönliche Daten

Name	Jens Garstka
Geburtsdatum	20.11.1975
Geburtsort	Bochum
Staatsangehörigkeit	deutsch
Eltern	Reinhild und Günter
Familienstand	verheiratet

Bildungsweg

Grundschule Witten-Herbede	08/1982 - 07/1986
Ruhr-Gymnasium Witten	08/1986 - 07/1995
Abschluß: Allgemeinen Hochschulreife	
Studium der Informatik an der TU Dortmund	10/1996 - 07/2009
Abschluß: Diplom-Informatiker	

Beruflicher Werdegang

Bundeswehr	10/1995 - 07/1996
Stud. Mitarbeiter, 1&1 ServiceLine	10/1996 - 08/2000
Stud. Mitarbeiter, AMG Consulting	09/2000 - 02/2001
Stud. Mitarbeiter, GETIT GmbH	03/2001 - 07/2009
Entwickler, GETIT GmbH	08/2009 - 04/2010
Wiss. Mitarbeiter, FernUniversität in Hagen, LG Mensch-Computer-Interaktion	05/2010 - heute