

STRUCTURAL INDEXING FOR OBJECT RECOGNITION

by

Fridtjof Johannes Stein

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA

In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Computer Science)

August 1992

Copyright 1992 Fridtjof Johannes Stein

Ich lebe mein Leben in wachsenden Ringen,
die sich über die Dinge ziehen.
Ich werde den letzten vielleicht nicht vollbringen,
aber versuchen will ich ihn.

Ich kreise um Gott, um den uralten Turm,
und ich kreise jahrtausendelang;
und ich weiß noch nicht: bin ich ein Falke, ein Sturm
oder ein großer Gesang.

Rainer Maria Rilke

I live my life in growing orbits
which move out over the things of the world.
Perhaps I can never achieve the last,
but that will be my attempt.

I am circling around God, around the ancient tower,
and I have been circling for a thousand years,
and I still don't know if I am a falcon, or a storm,
or a great song.

Acknowledgments

I thank my advisor and chairman of my thesis committee, Dr. Gérard Medioni, for all the support, advice and friendship throughout my studies at USC. He encouraged creative thinking, provided the necessary freedom, and directed me along the way towards the dissertation. I thank Dr. Ramakant Nevatia and Dr. Viktor Prasanna for finding time to serve on my thesis committee and providing invaluable suggestions during my qualifying exam.

Nothing of my work could have been accomplished without the support and friendship which I encountered at the IRIS group. Special thanks go to Dr. Keith Price, Andres Huertas, Dr. Steve Cochran, Hillel Rom, Gideon Guy, Yang Chen, and all the other members for various discussions, ideas, creative lunch breaks, and, of course, the steady flow of espresso.

I thank Dr. Michael Arbib and Dr. Aristides Requicha for their suggestions and encouragement at my qualification exam.

Some of the three-dimensional data was generated by Dr. Tin Jun Fan and Dr. Philippe Saint-Marc. Thomas Colvin and Dr. William Thompson at the University of Utah provided the panoramic images and the digital elevation map of the Wasatch National Forest.

I also want to thank Dorothy Steele, Kusum Shori, and Delsa Castello for their help and support over the years.

I express my sincere gratitude to Dr. Mike Brady for a wonderful Indian summer in Oxford, Great Britain, in 1986, when I spent several months at his department. This time was crucial for my further career because it triggered my interest in the field of object recognition and the correspondence problem in vision in general.

I especially want to thank my parents who invested so much time, thoughts, and money into my education. They always encouraged me to work towards a stay in a foreign country.

My deepest gratitude goes to my wife Hannelore who came with me across the ocean and supported me through all the ups and downs of my thesis. I also thank my little daughter Enja. Since her arrival eight months ago, life became very diversified, and while I was struggling to just teach the computer to distinguish between Mozart and a dinosaur, I was fascinated how she developed her vision with apparent ease.

Contents

Acknowledgments	iii
List Of Figures	vii
List Of Tables	x
Abstract	xi
1 Introduction	1
1.1 What is Computer Vision?	1
1.2 What is Object Recognition?	2
1.2.1 The Goal	4
1.3 What is Structural Indexing?	5
1.3.1 Limitations	6
1.3.2 Assumptions	8
1.4 Contributions of this Dissertation	9
1.5 Outline	9
2 Recognition of Two-Dimensional Objects from Two-Dimensional Images	11
2.1 Introduction	11
2.2 Related Work	11
2.3 Basic Idea	16
2.4 Recognition	18
2.4.1 Object Representation	18
2.4.2 Hypotheses Generation	18
2.4.3 Verification	19
2.5 Cardinality and Quantization	21
2.6 Results	22
2.6.1 Animal Shapes	22
2.6.2 Aircraft Shapes	23
2.6.3 Textured Shapes	23
2.6.4 Large Database	23

2.7	Limitations	34
2.7.1	Global versus Local Distortion	34
2.7.2	Detection of False Positives	34
2.8	Conclusion	35
3	Recognition of Three-Dimensional Objects from Three-Dimensional Images	37
3.1	Introduction	37
3.2	Related Work	38
3.3	Our Approach	43
3.3.1	The 3D Curve	43
3.3.2	The Splash	46
3.4	Robustness and Stability of the Splash Representation	54
3.4.1	Robustness in the Location	54
3.4.2	Robustness in the Reference Normal	55
3.4.3	Stability with respect to Noise	62
3.5	Recognition	64
3.5.1	Object Representation	64
3.5.2	Hypotheses Generation	65
3.5.3	Verification	67
3.6	Complexity Analysis	69
3.6.1	Hypotheses Generation	70
3.6.2	Verification	71
3.6.3	Summary	73
3.7	Results	73
3.7.1	Mozart	73
3.7.2	Plane and Wagon	74
3.7.3	Composers	74
3.7.4	Terrain Map	75
3.7.5	General Observations	75
3.8	Conclusion	88
4	Recognition of Three-Dimensional Objects from Two-Dimensional Images	89
4.1	Introduction	89
4.2	Related Work	91
4.3	Going from Edgels to Groupings	97
4.3.1	Preprocessing	100
4.3.2	Super Segments	100
4.3.3	Parallels	101
4.3.4	Symmetries	101
4.3.5	Closures	104
4.4	Proximity Indexing	106

4.5	Representation and Matching	109
4.6	Results	115
4.6.1	Buildings	115
4.6.2	Discussion	116
4.7	Conclusion	118
5	An Application: The Drop-Off Problem	122
5.1	Introduction	122
5.2	Related Work	123
5.3	Glossary	124
5.4	The Algorithm	127
5.4.1	The Basic Idea	127
5.4.2	Representation of a Horizon	128
5.4.3	Representation of the Map	131
5.4.4	Hypotheses Generation	133
5.4.5	Hypotheses Verification	134
5.5	Results	136
5.5.1	Results on Simulated Data	136
5.5.2	Results on Real Data	138
5.6	Conclusion	153
6	Conclusion and Future Research	154
6.1	Summary	154
6.2	Future Research	155
Appendix A		
	Least Square Method	156
Appendix B		
	Consistency between Hypotheses	158
	B.0.1 The Distance Constraint	158
	B.0.2 The Orientation Constraint	159
	B.0.3 The Direction Constraint	160
Appendix C		
	The Hashing Scheme	162
Appendix D		
	Skewed Symmetry Proof	164
Appendix E		
	Adjacency and Inclusion of Convex Approximations	166
	E.1 Adjacency	166
	E.2 Inclusion	167

List Of Figures

1.1	Algorithm for Structural Indexing	5
1.2	Example for Corrupted Grouping	7
2.1	Elephant Shape	16
2.2	2D Super Segment	17
2.3	Super Segment Eccentricity	17
2.4	Representation of a Model	19
2.5	Matching of Model(s) with Scene	19
2.6	Verification of Hypotheses	20
2.7	Animal Shapes	25
2.8	Hypotheses for the Elephant	26
2.9	Aircraft Model	27
2.10	Aerial Image of LAX	28
2.11	Edges of LAX	29
2.12	Detected Airplanes	30
2.13	Aquarium	31
2.14	Hypotheses for Textured Shapes	32
2.15	Large Database with Random Models	33
2.16	Discontinuity Distortion	34
2.17	Detection of a False Positive	35
3.1	3D Super Segment	44
3.2	Generation of 3D Curves	45
3.3	Splashes	47
3.4	n and n_θ	48
3.5	Vector Mapping	49
3.6	Interest Operator	53
3.7	The Corner Surface	55
3.8	Matched Splashes with Different Quantizations	56
3.9	Histogram of Angle Distribution	57
3.10	Example for $\Delta\phi(\theta)$ and $\Delta\psi(\theta)$	58
3.11	Envelope of Curve	59
3.12	Linear Approximation	60
3.13	$\Delta\alpha$	62

3.14	Matched Splashes with Different Quantizations	63
3.15	Stability Graph	63
3.16	Object Representation	65
3.17	Hypotheses Generation	66
3.18	Occurrence Distribution with respect to Cardinality	67
3.19	Verification	68
3.20	Data Base with 9 Objects	78
3.21	Example Mozart	79
3.22	Example Plane and Wagon	80
3.23	Hypotheses for Scene with Plane and Wagon	81
3.24	Three Composers	82
3.25	Projection of Detected Busts in Scene 3	83
3.26	Hypotheses found for Chopin	84
3.27	Shaded Terrain Map	85
3.28	Terrain Window	86
3.29	Result of Recognition (Scene projected on Model)	87
4.1	The Problem of Matching Local Features	97
4.2	Hierarchy	98
4.3	Multiple Interpretations	99
4.4	Example of a Parallel Symmetry	102
4.5	Skew Symmetry	103
4.6	Skewed Symmetries and the Corresponding Axes in Mozart Bust	104
4.7	Closures	106
4.8	Object 1: Convex Approximations, Graph and the Adjacency-Matrices	110
4.9	Object 2: Convex Approximations, Graph and the Adjacency-Matrices	111
4.10	Representation of a Model	113
4.11	Retrieval of Hypotheses from a Scene	114
4.12	Corresponding Subgraphs	115
4.13	Corresponding Subgraphs	116
4.14	Building Scene	119
4.15	Hypotheses Examples	120
4.16	Examples of Incorrect Hypotheses	120
4.17	Result	121
5.1	Horizon	126
5.2	Slice Through a Landscape at an angle θ	126
5.3	Map with Superimposed Reprojected Horizon 1	127
5.4	Graph of Horizon 1	128
5.5	Horizon Super Segment	129
5.6	Different Representations of a Map	131
5.7	Map with Superimposed Horizon Grid	132
5.8	Map Encoding	133

5.9	Hypotheses Generation	133
5.10	Hypothesis Errors	134
5.11	Hypotheses Verification	135
5.12	Approximations of Horizon 1	139
5.13	Hypotheses after Retrieval	140
5.14	Hypotheses for Horizon 1	141
5.15	Result 1	142
5.16	Simulated Panoramic Views for Horizon 1	142
5.17	Map with Superimposed Reprojected Horizon 2	143
5.18	Graph of Horizon 2	143
5.19	Hypothesis for Horizon 2	144
5.20	Result 2	144
5.21	Simulated Panoramic Views for Horizon 2	145
5.22	Map with Superimposed Reprojected Horizon 3	146
5.23	Graph of Horizon 3	147
5.24	Hypothesis for Horizon 3	147
5.25	Result 3	147
5.26	Simulated Panoramic Views for Horizon 3	148
5.27	View from Albion Campground	149
5.28	Shaded DEM Map of the Dromedary Peak Area	150
5.29	Result	150
5.30	Hypothesis for Result	151
5.31	Panoramic and Reconstructed View from Albion Campground	152
B.1	Distance and Orientation Constraints	159
B.2	Direction Constraint	160
D.1	Symmetries	164
E.1	Adjacency between CAs	166
E.2	Inclusion of CAs	167

List Of Tables

2.1	2D from 2D	15
3.1	3D from $2\frac{1}{2}$ D, 3D	42
3.2	Table of three-dimensional Objects	77
4.1	3D from 2D	96

Abstract

We address a major issue in computer vision, namely the problem of object recognition. An object recognition system, given a collection of object models and the sensory data about a scene, has to find instances of those objects in the scene. It has to find “what” is in the scene and “where”. This corresponds to the correct labeling of the scene and the estimation of the pose of the detected object models. We are addressing the problem in a realistic environment: the viewpoint is arbitrary, the objects vary widely in complexity, the library is large, and we make very few restrictive assumptions about the objects. Our general approach, called *structural indexing*, is novel in that it uses multiple representations and specific features. We show how these primitives can be encoded and stored into a table. This provides the essential mechanism for fast retrieval and matching. The scene features are used to retrieve hypotheses from the data base. Clusters of mutually consistent hypotheses represent instances of models. We study the different instances of the general problem.

- We present an algorithm for the recognition of flat objects. The models are acquired automatically and approximated by polygons with multiple line tolerances for robustness. Groups of consecutive segments (super segments) are then used as matching primitives.
- We designed a system which is able to recognize three-dimensional objects in range images. The novelty lies in the use of two different types of primitives for matching: small surface patches, where differential properties can be reliably computed, and lines corresponding to depth or orientation discontinuities. These are represented by *splashes* and 3D curves respectively.
- We suggest a solution to the “drop-off” problem, namely the localization of an observer in an unfamiliar environment. The approach shows how a specific three-dimensional problem can be solved by mapping it onto two dimensions.
- We propose an approach for the recognition of three-dimensional objects in a two-dimensional scene. As models, we use a set of non-registered views of a three-dimensional object. While most other systems use spatial relations for the recognition process, we use high level features and their topology.

Chapter 1

Introduction

1.1 What is Computer Vision?

As humans, we are capable of performing a huge variety of tasks based on the visual clues which we perceive from our environment. We orient ourselves by analyzing what we see around us. By using the information given by motion or the disparity between the two images seen by our eyes, we are able to estimate depth. We grasp and recognize objects. These mechanisms are found in most highly developed organisms. During evolution, the input sensor for the biological vision, the eye, was "invented" at least 38 times according to the zoologists Ernst Mayr and Luitfried von Salvini-Plawen [51]. The range of inventions spans the whole spectrum, from simple photoreceptor cells to very complex eyes with lens focus and color sensitivity capabilities, such as the human eye. This illustrates the importance for living creatures to deal with the information from visual data. It is a logical step in the evolution of computers and robots to also develop in this direction. The question is: how do we tap the visual resource?

During the last twenty years the range of applications has been widening, and it now includes many uses in manufacturing, medicine, navigation, and remote sensing. So far, most robots and actuators have been blind, relying solely on a well defined, rigid environment to perform their tasks. But, over time, these tasks are becoming too complex and the working environment too flexible. As a parallel development, the need for visual sensing and the processing of data is growing. Computer vision deals with these issues. It is defined as the science and technology of obtaining the necessary models, meanings, and control information from visual data.

In this dissertation we focus on one field in computer vision, namely object recognition. Object recognition is on one hand a fundamental part of computer vision, on the other hand, because of *labeling* visible objects, it links the pure technical aspect

of the input data with its semantic interpretation. This will enable future intelligent systems to deal with their environment on a very high symbolic level.

1.2 What is Object Recognition?

Human beings have no problem in recognizing objects. They are capable of picking up a tool from a cluttered environment, they recognize a certain building in a town, and they even recognize a person from a picture, taken from a view, from which they have never seen this person before. Furthermore, human beings are able to discriminate between thousands and thousands of objects. And when they see a newly designed chair, which they have never seen before, they classify it immediately based on its functionality.

What are the underlying principles? How does the discrimination works? How can human beings label their environment in order to understand it? In this thesis we do not try to find the biological justification behind this miraculous performance. We try to scratch the surface of the object recognition problem and look at it from an algorithmic point of view in order to design a system which can recognize objects based on vision. So, what is an object recognition system?

An object recognition system, given a collection of object models and the sensory data about a scene, has to find instances of these objects in the scene. It has to find *what* is in the scene and *where*. This corresponds to the correct labeling of the scene and the estimation of the pose of the detected object models.

The problem space involved in object recognition can be classified along several dimensions, formulated as questions:

What is a model? Models can be two-dimensional or three-dimensional. When we use the term *two-dimensional* in the following, we talk about objects which can be represented in a two-dimensional way, *e.g.* flat objects which can be represented by their boundary. The term *three-dimensional* corresponds to the three-dimensional spatial occurrence of an object. This can be represented by range data ($2\frac{1}{2}$ D), or by a CAD model, or by a set of different two-dimensional views, or by a set of two-dimensional or three-dimensional features.

We want to automate the process of object recognition. We therefore require the system to be able to directly acquire a model from the image without human interaction. This means that the object recognition system has to deal with models which might not be complete, *e.g.* because of self occlusion or erroneous low level processing.

What do we mean by a collection of models? Current research in object recognition often focuses on the pose-estimation part of the problem. The labeling problem is mostly ignored. This reduces the object recognition system to a geometric search process avoiding the problem of discrimination between a set of candidates. An object recognition system, as we understand it, has to be able to deal with more than one or two models. We are typically looking in the range of tens to hundreds of models in a model library. Several orders of magnitude above that, the human can deal with tens of thousands of models. To approach this performance is a future goal, which cannot be accomplished by state of the art systems.

What is the sensory data? Using real data from a sensor is always accompanied with noisy and quantized data. Perfect data cannot be expected. We also use the word "scene data" for the sensor data. As scene data we use gray level images obtained by a camera for two-dimensional recognition. For three-dimensional object recognition we use range data.

What is the matching strategy? In order to find the models in a scene we have to establish the correct correspondences. There are several ways to find the correspondences. The difference lies in the time complexity. Do we use a brute force method, such as exhaustive search? Do we try to constrain the search by reducing the search space? Or, as we propose, do we use indexing to reduce the search to a minimum?

How general are the objects? The generality of an object recognition system is conditioned by the generality of the objects it can handle. Most systems, including ours, restrict themselves to the subclass of non-articulated, rigid objects. Other assumptions can be made about the shape of the objects: Does an object have to be polygonal? Can the recognition system deal with curved objects? Is the recognition of objects such as fractals or statistical defined objects possible?

What kind of representation is used? How do we represent an object? Do we use its parts for the recognition? If yes, how do we segment the object into parts? Can we define an object in a generic way or do we need specific models?

Here we restrict ourselves to specific models. Using very local features, such as points and line segments results in a low discriminative power of the features. Focusing on more global object descriptions causes problems with the representation of sparse data and with the ability of the system to deal with occlusion. One of the main axes of our research is to develop features which make a compromise between

the local and the global descriptions. Smart grouping strategies allow to design specific features which are easy to match.

Do we have to make any assumptions about the viewpoint? This question is closely linked to the question of representation. We want the representation to be viewpoint invariant with respect to translation, rotation, and scale. The system should be able to deal with partial objects, which is necessary because of occlusion. Furthermore, we would like to handle deformations which are introduced through perspective projection.

What is the overall performance of the system? The performance of a system can be measured in different ways.

- The scene complexity is linked to the number of features in the scene.
- The data base complexity deals with the number of models stored.

Talking about complexity always includes *space* complexity and *time* complexity.

Summary: In all the above mentioned issues we have to deal with an inherent combinatorial problem, which has to be kept under control. The question comes down to: *How do we represent a large number of noisy, quantized, incomplete models in a way that allows us to match them efficiently against noisy, quantized sensor data?* We encounter in this question a typical trade-off: On one hand the representation has to be fine enough to discriminate between large numbers of (sometimes similar) models, on the other hand it has to be robust enough to overcome noise and quantization.

1.2.1 The Goal

Systems for model based object recognition have to address two components: representation of objects, and matching these representations. The goal of our work is to provide mechanisms for the efficient integration of these components at the software level. These two components are linked closely together. The choice of representation affects the matching. Therefore we focus our main effort on the representation.

Figure 1.1: Algorithm for Structural Indexing

We approach the correspondence problem with a mechanism called *Structural Indexing*. A rich source of information in visual data is the structure of the objects.

Structural Indexing tries to exploit this fact. The basic underlying algorithm is described as follows (see figure 1.1):

1. Find all structural token elements of all models and store them into a table.
2. Collect all the structural tokens of a scene.
3. Retrieve from the table all the tokens which are in the scene.
4. All corresponding tokens generate hypotheses.
5. The hypotheses have to be grouped with respect to constraints which assign them a membership to a model.

The questions which we address in future chapters are:

- What are these structural tokens?
- How are they stored in the table?
- How are they retrieved from the table?
- What are the grouping constraints?

All indexing schemes have in common that they use invariant feature characteristics on which they index. The invariance capability includes the handling of viewpoint changes, noise, and quantization.

1.3.1 Limitations

In the last section we talked about structural tokens, but we left several questions open. We shall discuss specific details in the different chapters of this thesis. Here we want to address the underlying problems which are inherent in the selection of features for the matching purpose.

In most aspects of our approach, we use edges as the main source of information. This is motivated by the fact that, on one hand, the object complexity is greatly reduced and, on the other hand, psychologists have found that boundaries in an image are extremely important. An object can often be recognized from only a crude outline of its boundary or pattern [19]. Given an image (of a model or a scene) we use an edge detector to obtain the edge primitives called *edgels*. These elementary primitives are very local and possess no discriminative power which would make them suitable for an efficient matching algorithm. We get our *elementary primitives*, the curves, by linking neighboring edgels. To further reduce the information and to

(e) Linked Corrupted
Edge Primitives

Figure 1.2: Example for Corrupted Grouping

obtain objects which are easy to process we approximate the curves with a linear approximation. This gives us a set of line segments which serve as building blocks for the higher level features.

An obvious way to proceed is to group the line segments with respect to certain grouping rules (*e.g.* linking of neighboring segments) in order to obtain higher level features which contain structural information corresponding to the applied grouping rules. To establish a match, a feature has to be computed in the model and another feature has to be computed in the corresponding area in the scene. But what happens if a rule can be applied for an area in the model and it cannot be applied to the corresponding area in the scene due to noise or inaccurate edgel primitives? This problem is illustrated in an example in figure 1.2. In figure 1.2(a) an image is given. A perfect edge detector would return edge primitives as shown in figure 1.2(b). A grouping of neighboring edges into curves would lead to curves (figure 1.2(c)). The junction could be detected at the point where the different curves meet. Figure 1.2(d) and (e) shows how corrupted edge data makes the detection of a junction extremely difficult, if not impossible.

There are several ways to counteract this problem. Relying on *several* representations simultaneously instead of only *one* representation is one possibility. We investigate two different flavors of this possibility:

- We emphasize in our work of recognizing two-dimensional objects from two-dimensional data (chapter 2) and three-dimensional objects from three-dimensional data (chapter 3) the representation based on more than one linear approximation. We focus on features which are extracted based on co-curvilinearity of line segments. This is a viable option when long connected edges can be extracted, such as in the silhouette boundary of a flat object, or in the representation which we developed to represent three-dimensional surface patches which guarantees to provide continuous curves.
- But what happens when we cannot rely on co-curvilinearity in the scene data? In the recognition of three-dimensional objects from two-dimensional data we address this problem. Continuity alone is too weak to serve as the only criterion to provide a good representation of an object. Instead of using only co-curvilinearity we expand the representation by using several grouping rules simultaneously. These grouping rules are based on the psychology literature in perceptual organization. They consist, among others, of strategies to detect occurrences of parallelism, closure, and symmetry.

1.3.2 Assumptions

When talking of models and objects, we try to be as much domain independent as possible. We, however, have to make some assumptions about the objects we deal with:

- In this dissertation we restrict ourselves to rigid objects.
- Because in most of our approaches we use edges as the main source of information, we assume that light is reflected by an object in a way that a state-of-the-art edge operator (we use the Canny edge detector [16]) detects the edges of the object. However, these do not have to be perfect for our systems to work.
- We assume further that the sensor data is dense. Due to the nature of our features we cannot work with sparse data.

1.4 Contributions of this Dissertation

We have developed several original ideas which we list in the following:

- We have developed novel features for the recognition of two-dimensional and three-dimensional objects.
- We have implemented object recognition systems for two-dimensional and three-dimensional objects that can handle large sets of objects.
- We have designed an algorithm which presents an application to a navigational problem.
- We propose a novel algorithm for the recognition of three-dimensional objects from two-dimensional images.

1.5 Outline

Object recognition can be performed at different levels of complexity and difficulty. This is reflected in the structure of this dissertation. We start with the easier two-dimensional problem and proceed to higher dimensions. In addition, the order in chapters 2 to 4 corresponds to the historical evolution of this thesis. The idea of structural indexing is crucial to all proposed algorithms.

We start in chapter 2 with the recognition of flat objects. We first give a survey of the previous work. We then present our algorithm and illustrate it on several examples.

Chapter 3 deals with the extension of the two-dimensional work to three dimensions. We start by looking into the research that was done in the field so far and discuss previous accomplishments. The presentation of our algorithm includes the introduction of a new feature, which we call the *splash*. It allows us to find correspondences between free-formed surface patches. A robustness analysis, a complexity discussion and various results are presented.

In chapter 4, we focus on the recognition of three-dimensional objects from two-dimensional data. The first section is a review of related work. We then skim the topic of *perceptual organization* for the development of a feature hierarchy. A novel algorithm is introduced, allowing us to match topological graphs.

An applied algorithm is presented in chapter 5. The approach shows how a specific three-dimensional problem can be solved by mapping it onto two dimensions.

The concluding chapter contains a summary and a brief discussion of proposed directions for future research.

In the appendices we elaborate on issues which did not fit into the flow of the dissertation, such as proofs and algorithmic details.

Most of the work described here has been presented in [78–85].

Chapter 2

Recognition of Two-Dimensional Objects from Two-Dimensional Images

2.1 Introduction

The research in two-dimensional object recognition is a specific topic in the overall problem. It includes all tasks where the recognition of flat objects is required, such as some industrial applications, and aerial image analysis. In this chapter we start with a survey of related work. We present our approach and discuss some results. An extensive robustness and complexity analysis of the proposed algorithm can be found later in chapter 3, in sections 3.4 and 3.6 respectively.

2.2 Related Work

By talking about “two-dimensional from two-dimensional ” recognition we are not interested in the broad field of template matching. Template matching is mainly concerned with finding non-occluded, non-rotated objects with fixed scale. The general tool to do template matching are correlation based algorithms. We instead address the issues of viewpoint invariance which include rotation, scale, occlusion, and perspective skewing. When we talk about two-dimensional objects, we are concerned about general two-dimensional silhouettes.

Bolles and Cain: Bolles and Cain [9] describe the local-feature-focus method, which is an algorithm to recognize and locate partially visible two-dimensional objects. First the image is scanned to detect local features. In the example presented in their paper, three types of features are detected in the binary image: round holes, convex 90° corners and concave 90° corners. The position of each feature is recorded, along with its size and orientation. When each model is being acquired,

the algorithm searches for a cluster of local features in a relative configuration that does not occur elsewhere in the model or in any other possible model. One feature in the cluster is selected as the “focus” feature. When searching for this model in the image, the focus feature is first searched for. If it is found, its neighborhood is searched for the remaining features in the cluster. If they are found, and their relative configuration is consistent with the configuration found in the acquisition stage, the model is hypothesized to exist at the location. A template for the model is then translated and rotated by the required amounts and is matched to the image. The method works fast for models with significant features, such as the examples given in the paper. For the general case, the method has the problem of detecting significant features. Unique features may be difficult to find.

Grimson and Lozano-Pérez: Grimson and Lozano-Pérez [34, 35] describe a system which is able to recognize objects from sparse scene data. If there are m known objects with n_j segments each and s scene segments, there are $\sum_{j=1}^m (n_j)^s$ combinations of pairings between scene and model segments. The system tests these combinations using a tree search. Not all these combinations need to be tested because distance and angle constraints are used to prune almost all the combinations. Still, the number of combinations that need to be tested grows rapidly with object complexity. For those combinations not ruled out, the system tries to find an object position and orientation consistent with the segment assignments. If a consistent transformation is found, the object is recognized. In [32], Grimson shows that, under some simple assumptions, the expected complexity of recognizing isolated objects is quadratic in the number of model and scene fragments, but that the expected complexity of recognizing objects in cluttered environments is exponential in the size of the correct interpretation.

Knoll and Jain: A smart approach based on some heuristics and under the assumption of fixed scale is described by Knoll and Jain [45] in their paper, which is based on what they call *feature indexed hypotheses*. They take advantage of the similarities and differences between model types to group candidate models. Instead of unique features, features common to several models in the model set are used. For each feature, a list is kept of where it occurs in each object type. When a match is found for a feature in an image, models are hypothesized for each object identity and orientation in the feature’s list. Each of these hypotheses is then tested using a template match to determine which, if any, are correct.

Schwartz et al. : A system which is able to recognize models from a database with up to 100 models was introduced by Kalvin, Schonberg, Schwartz and Sharir

[42]. They assume fixed scale and concentrate their representation effort on the segmentation of the boundary in boundary parts, which are likely to belong to one model, and which they call *footprints*. These footprints are used to match a scene against a database with a hashing scheme based on some heuristics. They call this indexing mechanism *geometric hashing*. The indexing allows them to reduce the search for a model in a scene to a small set of possible candidate models, which furthermore allows them to use a large model database.

HYPER: The method described by Ayache and Faugeras [3] in the system HYPER uses polygonal approximation for the representation of scenes and models. In an iterative way the model segments and scene segments are matched until certain quality thresholds are reached. For the cluttered scenes presented in their paper they show very good performance both in recognition time and in accuracy.

Tucker et al. : A parallel implementation of object recognition in two dimensions is done by Tucker, Feynman and Fritzsche [92]. They assume a fixed scale. Models and scenes are represented as boundary features, which are used for the generation of hypotheses by matching scene and model features. Parameter space clustering of hypotheses is used to rank hypotheses according to preliminary evidence prior to verification. The authors show experimental results using databases containing between 10 and 100 object models.

Cass: Another parallel implementation on a connection machine is described by Cass [17]. Object model and scene data is represented as contour features. A method called *transformation sampling* is used to determine the optimal model feature to scene feature transformation by sampling the space of possible transformations. He shows that only a small part of this space need actually be sampled due to the constraints placed on possible transformations by individual matches of scene features to model features.

Lamdan et al. : Another method based on indexing was suggested by Lamdan, Schwartz and Wolfson [48, 49]. They have developed an algorithm which consists of two steps:

1. First they preprocess the model. They use the so called *interest points* of a model (like deep concavities, sharp convexities, ...) to build a table. For each ordered non-collinear triplet of model points the coordinates of all other model points are computed taking this triplet as an affine basis of the 2-D plane. Each

such coordinate (after quantization) is used as an entry to a table, where the pair (basis-triplet and model) is recorded.

2. In the recognition stage the interest points of the scene are extracted. An arbitrary ordered triplet in the scene is chosen and the coordinates of the scene points taking this triplet as an affine basis are computed. For each such coordinate as an index, the appropriate entry in the table is checked and for every pair recorded there a vote is tallied. If a certain pair scores a large number of votes, the decision is made that this triplet corresponds to the one chosen in the scene. The uniquely defined affine transformation between these triplets is assumed to be the transformation between the model and the scene.

The system deals successfully with the combinatorial explosion of possible interpretations inside a model by using the scene coordinates as an index for a voting scheme. But if we have to deal with multiple models and cluttered scenes with a lot of extra points, the system suffers from ineffective search through all the possible triplets. In the worst case, when the model we are searching for is *not* in the scene, we have to check all triplets first.

In a more recent paper [4] the approach was extended to deal with articulated objects, consisting of rotational and translational joints.

Ettinger: Ettinger [25] introduces a system that exploits hierarchies of object structure and object scale (coarse to fine). He focuses his attention to use these hierarchies to achieve robust recognition based on an organization that allows him to use an indexing scheme for his model library. He develops an object shape representation that incorporates a component subpart hierarchy. These subparts allow mutual relative parametrization and they are the building material for the model library. His implementation uses a representation based on significant contour curvature changes. Based on his paper, it is not clear how well his system copes with occlusion.

Califano and Mohan: Califano and Mohan [15] propose an indexing algorithm composed of a two stage process. First, short-range autocorrelation operators are used to map from the image pixels into a small set of simple localized shape descriptors. A global autocorrelation operator then maps combinations of these local descriptors into invariant indices. The indices are used to address the cells of a global lookup table which contains the shape model representations. They show extensive tests with a data base of up to 300 arbitrary shapes. They demonstrate the fault tolerance of their system by systematically destroying random portions of the shape memory and then evaluating recognition performance.

Table 2.1: 2D from 2D

2.3 Basic Idea

For two-dimensional objects it is natural to use boundaries as a representation. Our representation of a model or a scene is based on polygonal approximations. We are not dependent of any feature detection algorithm and we do not have to handle explicit distinguished points like corners or inflection points. The polygonal approximation of a curve captures some of the curvature information in the form of the angle between consecutive segments. These curvature angles are invariant with respect to scale, rotation and translation. Because we store only a coarsely quantized value for these angles, our system exhibits a significant tolerance to “weak perspective” as well.

Obviously, the polygonal approximation for a curve is not unique (see for example figure 2.1). It depends on the line fitting tolerance. Therefore, for the purpose of robustness, we use *several* polygonal approximations with different line fitting tolerances simultaneously. In addition, this allows us to compare objects of different scale. Since we want to handle occlusion, we do not expect to obtain complete boundaries in our scenes, but only portions of them. On the other hand, individual segments are too local to be useful as matching primitives. Grouping a fixed number of adjacent segments provides us with our basic features, the super segments. In order to manipulate and to encode super segments we have to define some terms and to describe some attributes (see figure 2.2):

cardinality The cardinality of a super segment is the number of segments it consists of.

arclength The arclength of a super segment is the sum of the segment lengths.

angles Let a super segment consists of n segments, then we have $n - 1$ angles between successive segments.

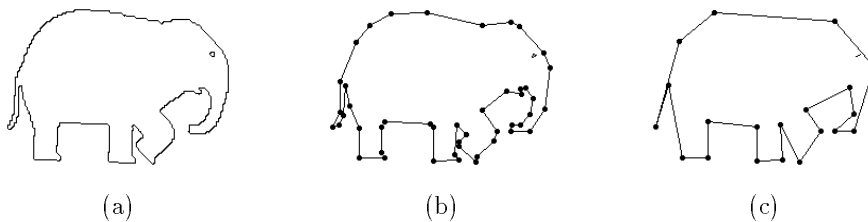


Figure 2.1: Elephant Shape (a) and Different Polygonal Approximations (b) and (c)

(b) super segment 2

Figure 2.3: Super Segment Eccentricity

represented as the ellipse which can be computed from the covariance matrix of the vertices. The two eigenvectors of the covariance matrix define the axis of the ellipse. We use the eccentricity (ratio of the length of the small and the long axis) as a feature for super segments. In figure 2.3 we show an example of two super segments which have the same angles but a different eccentricity.

As mentioned before, we are mainly interested in the curvature information implicitly captured by the super segment angles. That is the reason why we use them to encode a super segment. The eccentricity measure allows us to distinguish between super segments which have the same angles but totally different shapes.

We encode a super segment ss with cardinality n in the following way: The list of the quantized curvature angle values and the quantized eccentricity is the code of the super segment ss :

$$\text{Code}(ss) = (\text{Quant}(\alpha_1), \text{Quant}(\alpha_2), \dots, \text{Quant}(\alpha_{n-1}), \text{Quant}(\text{eccentricity}))$$

All the encoded super segments serve as keys into a table (the data base), where we record the corresponding super segments as entries, as explained later.

2.4 Recognition

2.4.1 Object Representation

As mentioned in the previous sections, we represent our model (or scene) with super segments. Therefore we first apply an edge detection algorithm on the image containing the model (or scene). We use the Canny edge detector [16] for grey level images and a simple boundary tracer for binary images. The resulting edgels are further processed with a line fitting algorithm to compute the polygonal approximations. Connected linear segments form chains of adjacent segments. The segment chains provide the super segments by grouping a fixed number of adjacent segments. We then take all the super segments, encode them and use the code as a key for a table, where we record the super segment as an entry (see figure 2.4). The table is implemented as a hash table. This allows efficient storage and retrieval (see appendix C).

2.4.2 Hypotheses Generation

By using indexing for the hypotheses generation process, we select a set of candidate models that are likely to be present in the image. The scene is preprocessed as explained in section 2.4.1 to generate all the super segments. These are encoded

Figure 2.5: Matching of Model(s) with Scene

and the corresponding codes are used as keys to retrieve the matching hypotheses between the super segments of model and scene (see figure 2.5).

2.4.3 Verification

We compute all possible matches for the super segments of the scene with the model super segments to generate multiple hypotheses. Next, we divide these n hypotheses $H = \{h_1, h_2, \dots, h_n\}$ according to which model the model super segment of the hypothesis belongs to. We store these into a correspondence table where we have the models m_i as keys and the i_k hypotheses $H_i = \{h_1^i, h_2^i, \dots, h_{i_k}^i\}$ (with $H_i \subseteq H$) as entries (see figure 2.6). The next step is the formation of consistent clusters. For every model m_i we have to check which hypotheses h_x^i and h_y^i with $h_x^i \neq h_y^i$ are consistent with every other. We do not check every hypothesis against each other, instead, we adopt the criterion that three consistent hypotheses are sufficient to instantiate the model in the scene. If we have three consistent hypotheses $C = \{h_r^i, h_s^i, h_t^i\}$ with

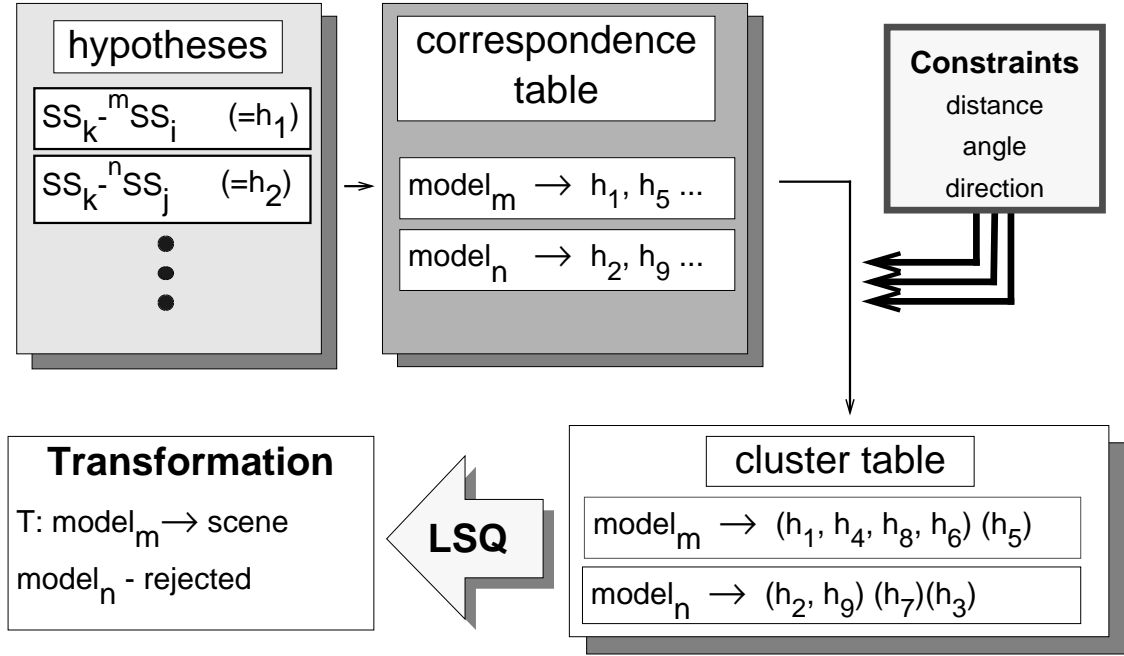


Figure 2.6: Verification of Hypotheses

$C \subseteq H_i$ for one model m_i , we examine the remaining hypotheses in $H_i \setminus C$ and collect those that are consistent with at least one of the selected three in C . When we have found one instance, represented by $I = C \cup F$, with F the additional found consistent hypotheses, we try to find more instances in the remaining hypotheses $H_i \setminus I$.

But what is meant by consistency? We use the powerful constraints introduced by Grimson and Lozano-Pérez [34, 35] to prune efficiently the interpretation trees, and build our clusters. These constraints are as follows:

distance The range of distances between the locations of two super segments in the scene must lie within the range of distances of the corresponding model super segments. To compensate for scaling differences, we use the ratio of the arc lengths of the super segments as a scaling factor.

angle The angle between the orientation vectors of two scene super segments must lie within the range of angles of the corresponding model super segments.

direction The range of components of a vector spanning the two scene super segments in the direction of each of the super segment's normal vector must lie

within the corresponding range of super segments for vectors spanning the model super segments.

In the two dimensional domain, these three constraints define the attitude of one feature relatively to another since it specifies all degrees of freedom.

After we group the hypotheses into clusters which represent instances of models, we can compute the transformation from the model coordinates to the scene coordinates by applying a least squares fit to all the matching super segments (see appendix A). Because of noise, we generally get a good first guess for the transformation, but not an exact match. A second least squares match on corresponding corners or segments can refine the result. This is similar to the refinement procedures used by Lamdan [48, 49], Huttenlocher [38, 39], and Mundy [58].

2.5 Cardinality and Quantization

Which super segment cardinalities should we use in our system? To use a fixed cardinality is possible, but it reduces the flexibility of the matching process. But when we have long super segment matches, why not use them? Therefore we compute all possible cardinalities in the range from 3 to 6. That means that a curve which is approximated by six linear segments is represented by a super segment of cardinality 6, two super segments of cardinality 5, three super segments of cardinality 4, and four super segments of cardinality 3. Higher cardinalities of matched super segments increase the probability of having a *good* match. In the case of gray level images, where we have no directional information we use double the amount of super segments, one for every direction (see the example in section 2.6.2).

A crucial parameter for the hypotheses generation is the quantization size for the encoding of the super segments. Two super segments match, when both keys are exactly the same. This means that all the pairwise values have to fall into the same quantization intervals. Several questions have to be raised:

- Which quantization (interval size) is the best?
- Should we use different quantizations for different cardinalities?

To answer the two questions, we have to look at the probability of a match, given a certain key length n . We assume equal quantization for all keys. Suppose the range is quantized into intervals of the the same size q . Each value, v , is then assigned a key based on which interval v falls into. If the value v is corrupted by a random additive term bounded by ε , the probability that $v + \varepsilon$ is assigned the same key k as v is simply

$$p(k|v) = 1 - p(k - 1|v + \varepsilon) - p(k + 1|v - \varepsilon) = 1 - \varepsilon/q.$$

When we have a vector of such values v , of length n , the probability that the corrupted vector is assigned the same entry as the original one is

$$p^n(k|v) = (1 - \varepsilon/q)^n. \quad (2.1)$$

Looking at this equation, it is obvious that the same quantization for different keys increases the probability for matches of small cardinality and decreases the probability for matches of large cardinality. In our implementation we try to counteract this effect by using larger interval sizes for larger cardinalities. Typical values are (only for the quantization of the curvature angles):

cardinality	3	4	5	6
interval size	30°	40°	45°	60°

For extremely noisy data we use slightly larger values, for data with little noise we use smaller values. The choice of a larger quantization than necessary is not critical, as the system will detect the same objects. However, due to the increased number of retrieved hypotheses, recognition is slower (see the discussion in section 3.6).

2.6 Results

2.6.1 Animal Shapes

We have tested our system with a library of twelve animal shapes (see figure 2.7(a)). We obtained them from coarsely digitized binary images. We look for objects with a large variety of features. We want simple objects, complex objects, and objects with a lot of similarities. Scene 1 was taken by printing the three animals, enlarging them, and cutting them out. Finally we put the three silhouettes on a light table and took a picture with a CCD camera. The camera had an angle of about 20 degrees to the normal of the light table. This procedure guarantees scaling, occlusion, rotation, translation, weak perspective, and noise (thanks to our high skills in cutting out - look at the ears of the giraffe). Scene 2 was composed by synthetically overlaying a moose, a scaled moose, and a deer. The recognition time (not including representation generation time) for scene 1 (see figure 2.7(b) and (c)) and for scene 2 (see figure 2.7 (d) and (e)) was both times below 10 seconds on a Symbolics 3675 lisp machine. To illustrate what kind of hypotheses are found by the system we show in figure 2.8 some of the hypotheses which led to the detection of the elephant in scene 1.

2.6.2 Aircraft Shapes

In another example (see figure 2.9, 2.10, 2.11, and 2.12) we try to recognize airplanes from an aerial photograph of an airport. The database consists only of one airplane model (see figure 2.9). We create the polygonal approximations for the airplane model manually. The scene is displayed in figure 2.10 and the corresponding edges in figure 2.11. The recognition time (not including representation generation time) for the nine planes in the scene is approximately 70 seconds. The detected aircraft shapes are displayed in figure 2.12. All but two of the large aircrafts are detected. This shows the limit of our system in dealing with inherently noisy edge data. When the edge data is severely corrupted by shadows, connected super segments cannot be found.

In this example the verification step requires relatively more time than in the animal example. The reason is that we have several instances whose hypotheses all have the same model index. To reduce the computation time, we include an intermediate indexing step. Based on the fact that super segments which belong to one airplane in the scene have similar transformation parameters, we use the translational part to preindex the hypotheses.

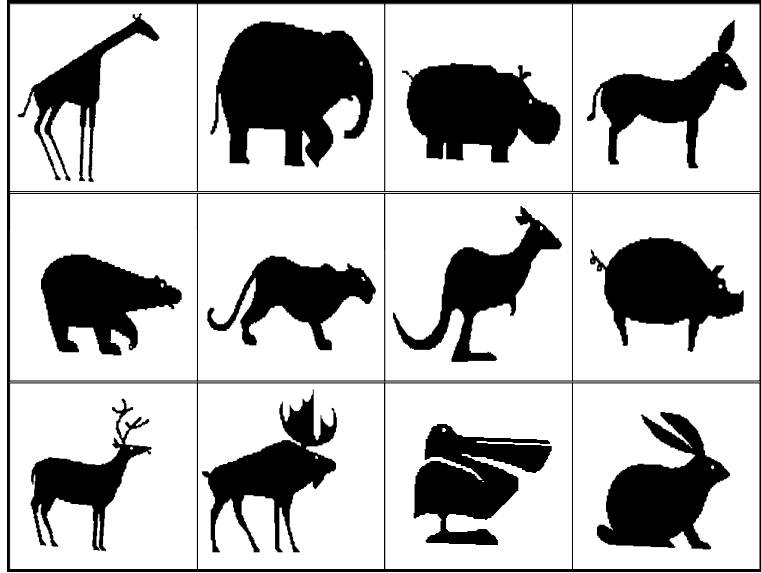
2.6.3 Textured Shapes

Our recognition algorithm can also be used for the detection of textured flat objects. This is illustrated with an example in figure 2.13. The models are shown in figure 2.13(a) and the corresponding edges in figure 2.13(b). We try to find the fish shapes in the scene in figure 2.13(c) (for the corresponding edges see figure 2.13(d)). The detected shapes are shown in figure 2.13(e). Instead of just using the outlines of the objects, the program uses in addition the “inner edges”. A few examples of hypotheses can be seen in figure 2.14.

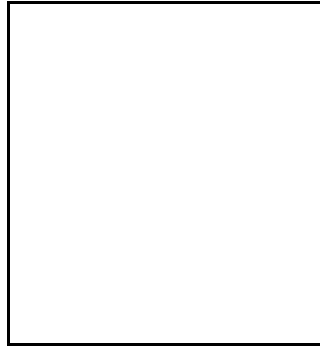
2.6.4 Large Database

In this example we direct our attention to a large database (see figure 2.15). We created 200 models by randomly overlapping more than 3 and less than 7 random triangles. Our models consist, in the average, of 48 super segments. Our scene is generated by taking three models out of the 200. We rotate and overlap them artificially. Then we let the system recognize the scene with different database sizes. The results are shown in the graphs in figure 2.15. There are several interesting observations:

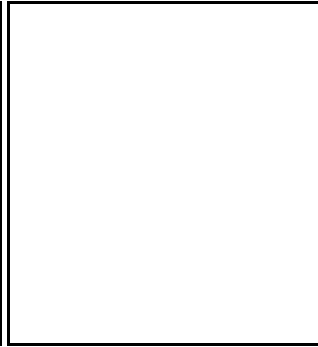
- The table does not grow linearly. There is a certain saturation effect. The cause is the nonequal distribution of the (encoded) super segments in the table.
 - Super segments in a random scene occur with different probabilities: a super segment consisting only of angles between +120 and +180 degrees is much less likely than a super segment with angles between 0 and +60 degrees.
 - By generating models based on a limited number of triangles, we constrain the super segments to a certain subset of all possible super segments.
- The recognition time curve has in its linear range the slope of approximately 0.1 seconds per model. The saturation effect results in an increasing steepness. The cause is the increase of the average number of hypotheses per table entry.
- The first three models in the data base are the ones present in the scene. Therefore the recognition time increases very fast for the first three models. The slope is approximately 1.3 seconds per model. Adding more models to the database has only little effect on the performance. The time to recognize the three models in the scene with a database containing just these three models takes about 4 seconds. However, recognizing the same scene with a database containing 50 models takes about double the time, only 8 seconds.



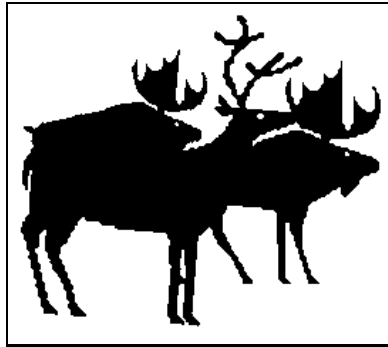
(a) Data Base



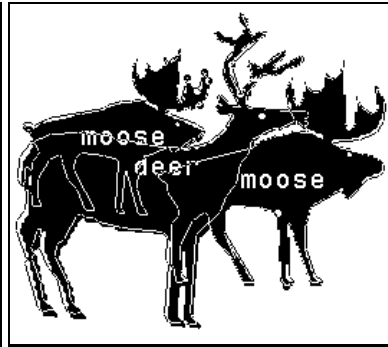
(b) Scene 1



(c) Detected Animals



(d) Scene 2



(e) Detected Animals

Figure 2.7: Animal Shapes

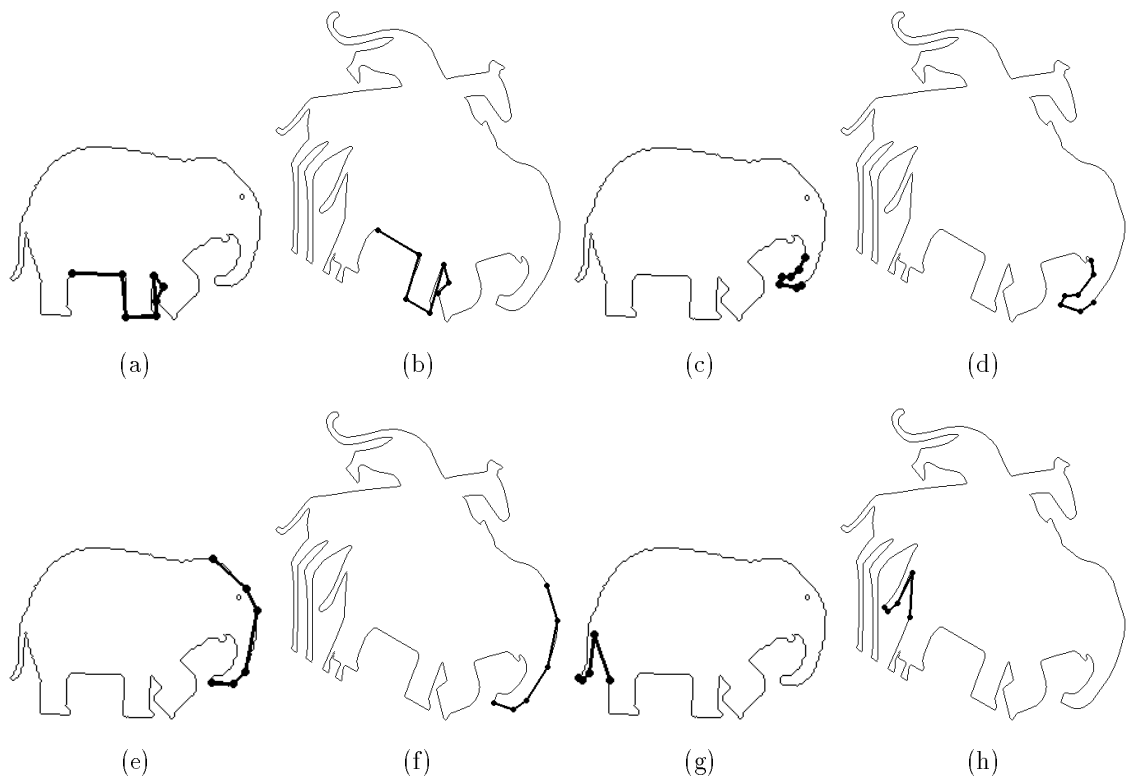


Figure 2.8: Example for Hypotheses for the Elephant - (a,b): Hypothesis 1, (c,d): Hypothesis 2, (e,f): Hypothesis 3, (g,h): Hypothesis 4

Figure 2.9: Aircraft Model



Figure 2.10: Aerial Image of LAX

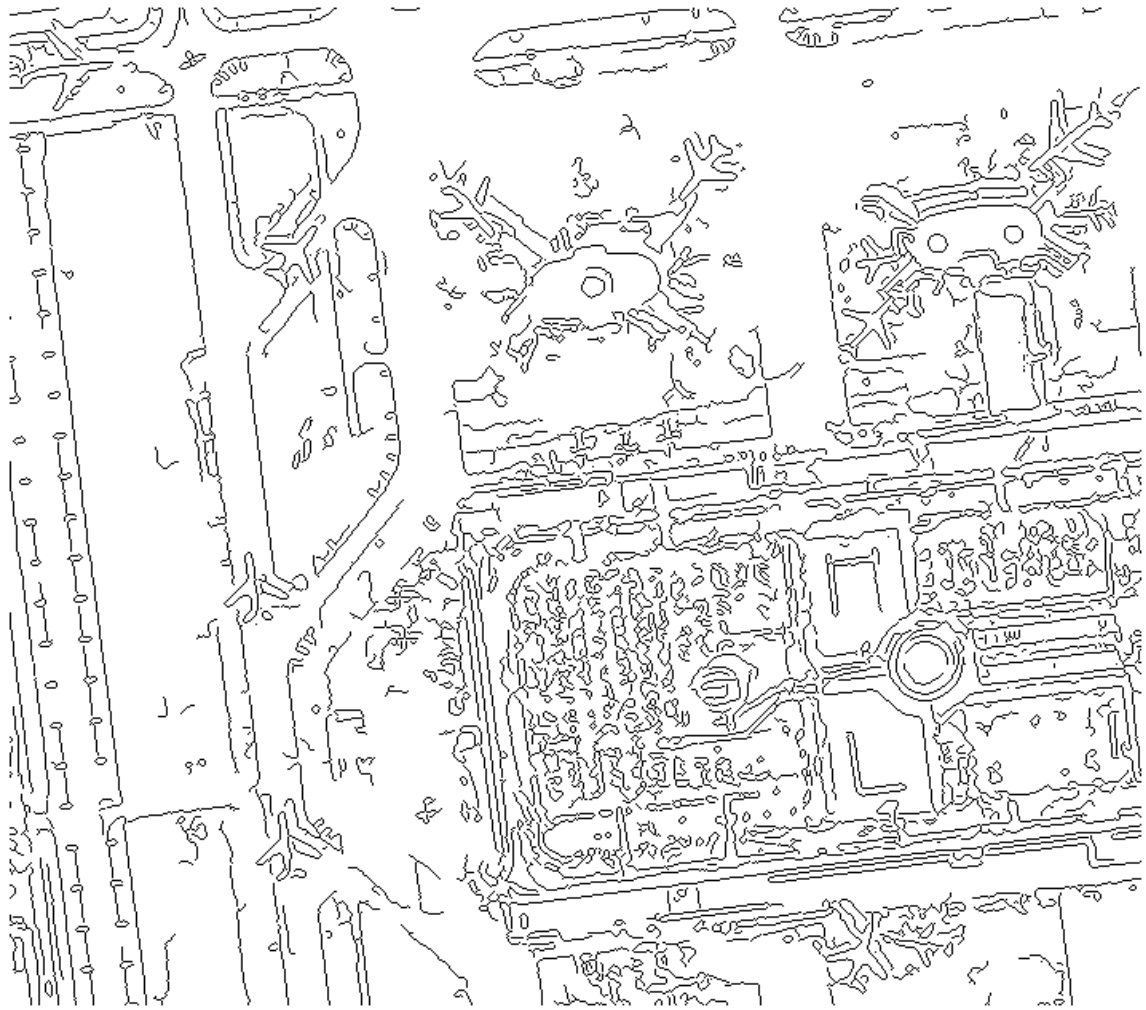


Figure 2.11: Edges of LAX



Figure 2.12: Detected Airplanes

(b) Corresponding Edges

(c) Fish Soup Scene

(d) Corresponding Edges

(e) Detected Models

Figure 2.13: Aquarium

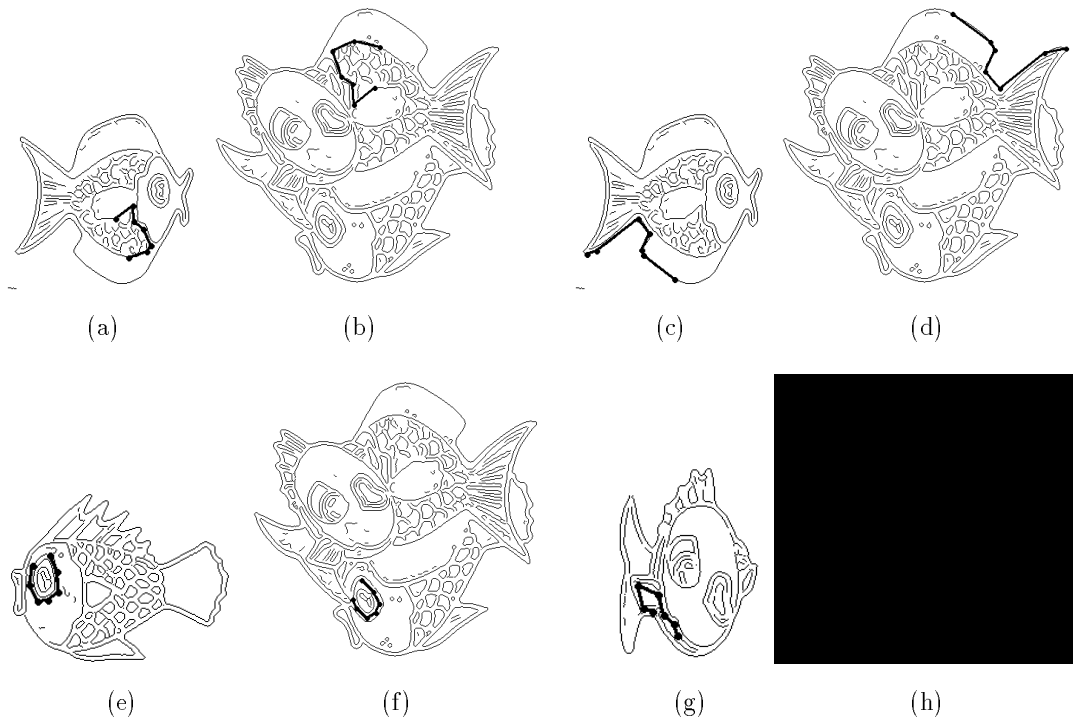


Figure 2.14: Examples of Hypotheses for Textured Shapes - (a,b): Hypothesis for fish 1, (c,d): another Hypothesis for fish 1, (e,f): Hypothesis for fish 2, (g,h): Hypothesis for fish 4

Figure 2.15: Large Database with Random Models: (a): Recognition Time, (b): Size of Table, (c): Number of Hypotheses

(c) Global Distortion

Figure 2.16: Discontinuity Distortion

A representation which would be able to handle such a recognition task has to use very local features, such as points. Grimson's interpretation tree approach or Lamdan's geometric hashing would possibly still be able to recognize the elephant figure 2.16(c) (see section 2.2).

2.7.2 Detection of False Positives

A goal of a reliable object recognition system is to detect as much objects as there are in the scene, but not more than that. But sometimes there are several possible interpretations of the underlying data. For example, in figure 2.13 the eyes of all the

Figure 2.17: Detection of a False Positive

fish are very similar. Therefore the system finds very strong support for an object as shown in figure 2.17. How do we exclude such a mismatch? In our current implementation we resolve conflicts based on the amount of the supporting hypotheses for an object instance. We do not consider whether these hypotheses are all located very local (as in the example in figure 2.17). If two instances share a scene super segment, the instance which is supported by more hypotheses wins, and the other instance is discarded. This is a heuristic which is not guaranteed to always work.

A better approach would be to collect all instances and to process them by a higher level system which would decide which instance is likely to represent a correct detection. The decision could be based on measures such as the coverage of the image area and the supporting hypotheses based on local *and* global considerations.

2.8 Conclusion

We have presented an algorithm which is able to recognize two-dimensional shapes from a two-dimensional scene. We have demonstrated the performance of on different objects, such as animal silhouettes, airplanes, textured, and random objects. The results lend support to the claim that indexing is a powerful tool, and that multiple representation provides robustness. These properties allow our system to function efficiently in the presence of noise and occlusion. In addition we have discussed the limitations which are inherent in the choice of our basic feature, the super segment.

To overcome these limitations in a future system, the representation scheme has to be extended to include other grouping strategies, such as parallelism, or symmetry.

Chapter 3

Recognition of Three-Dimensional Objects from Three-Dimensional Images

3.1 Introduction

In this chapter we present an object recognition system which is able to match general three dimensional objects from partial three-dimensional data in an efficient way by using *structural indexing*. By talking about “general objects” we make very few restrictive assumptions about their shape, so we only exclude statistically defined shapes (e.g. foams) and crumpled objects (e.g. fractals) [46].

Representing a three dimensional solid object is either possible by using a surface or a volumetric description. Volumetric descriptions from a single view require a difficult inference step to compensate for the unseen part, so we will use descriptions based on visible surface instead. The task of three-dimensional object recognition involves identifying a correspondence between a part of one range image and a part of another range image with a particular view of a known object. This requires the ability to match one feature of one range image against a feature of another range image. A feature can be either a surface patch or a general 3D curve. The question is: “How can we represent such features so that they can be matched in an efficient way?”

This chapter is organized as follows: Section 3.3 presents our approach to recognize objects in range imagery. We focus on the choice of our two features:

The 3D Curve For some objects, such as polyhedra, it is natural to use a representation based on edges. For that reason we introduce in subsection 3.3.1 our basic feature for the representation of surface and depth discontinuities, the *3D curve*. We achieve a robust and stable representation by using multiple line fitting tolerances to obtain a set of polygonal approximations. The polygonal approximations are grouped in sets of connected segments. These super

segments are encoded based on the angles between consecutive segments, providing invariance with respect to rotation, translation, and scale (even though scale is not needed in three dimensional object recognition).

The Splash For some objects, however, such as objects bounded by free form surfaces, it is difficult to use edges for the representation. Therefore, in subsection 3.3.2, we present a new representation, the *splash*, based on small surface patches where we can compute differential properties in a reliable way. A splash consists of a radial decomposition of surface normals. It is a local Gaussian map describing the distribution of surface orientation along a geodesic circle. A splash can be represented by two two-dimensional periodic functions, which can also be combined into one, compact, three dimensional curve. This allows us to use a unified representation scheme for the splash *and* the above mentioned 3D curve.

In subsection 3.4 we address in detail the issues of robustness and stability of these two features. In subsection 3.5 we describe how we use a table to store our features and retrieve them efficiently for hypotheses generation. We provide a complexity analysis in section 3.6 with respect to the number of models in the data base and with respect to the scene complexity. We present two experiments which we performed to analyze the behavior of the TOSS system for large data bases (100 models). In section 3.7 we present some results on real data.

3.2 Related Work

Reviewing the existing systems, none so far (known to the author) is able to represent, match, and recognize *general* three dimensional objects. An excellent overview regarding the problem of matching free-formed surfaces can be found in [6]. Most object recognition systems to date either rely on exact, CAD-like models, or make restrictive assumptions on the possible shape of the surface patches. The following is a summary of related work.

Nevatia and Binford: Nevatia and Binford [63] developed a system which can recognize curved objects from range images. They present a technique for generating structured, symbolic descriptions of objects by segmenting them into simpler sub-parts. For the description they use generalized cones. The recognition is performed by matching the scene description with stored model descriptions. Because of the segmentation into sub-parts, they can handle articulated objects.

Grimson and Lozano-Pérez: See section 2.2.

Bhanu: Bhanu [7] presents a 3D scene analysis system for the shape matching of real world 3D objects. Object models are constructed using multiple-view range images. At first, range images of each model object from different views are acquired. The relative orientation between each pair of views is assumed to be known. Then the surface points of the object in each view are isolated from the background by removing pixels with large depth values (far away from viewer). Second, a large number of object surface points are obtained by transforming points from each individual views into a common object-centered coordinate system. Finally, the object is represented as a set of planar faces approximated by polygons. This is accomplished by a two-step algorithm. In the first step, a three-point seed algorithm is used to group surface points into face regions, and in the second step, the face regions are approximated by 3D planar convex polygons.

Shape matching is performed by matching the face description of an unknown view with the stored model using a relaxation-based scheme called *stochastic face labeling*. The face features obtained above are used to compute the initial face-labeling probabilities for possible matching between each model face m_i and each face in the unknown view s_i . Then the labeling probabilities are updated by checking the compatibilities in geometric transform for each pair $\langle m_i, s_i \rangle$. The compatibility between m_i and s_i is obtained by finding the transformation T_i between them, and applying T_i to other pairs $\langle m_j, s_j \rangle$, computing the error in feature values after the transformation. The match is less compatible if the error is large.

3DPO: Horaud and Bolles [66] and Bolles *et al.* [10] present the 3DPO system for recognizing and locating 3D parts in range data. The model consists of two parts: an augmented CAD model and a feature classification network. The CAD model describes edges, surfaces, vertices, and their relations. The feature-classification network classifies observable features by type and size. The model objects are represented by a tree-like network such that each feature contains a pointer to each instance in the CAD models. In the range images discontinuities are detected and classified into cylindrical and linear curves. A local-feature-focus method is used for the matching process. At first, the system searches for features that match a feature for some model, for example, a cylindrical curve with a given radius. Then objects are hypothesized by determining whether a pair of observed segments are consistent with a given model feature. The system shows good results on bin-picking tasks of industrial parts.

Faugeras and Hebert: Faugeras and Hebert [28] developed a system to recognize and locate rigid objects in 3D space. Model objects are represented in terms of linear features such as points, lines, and planes. Range images are used as input. The same features such as significant points, lines, and planes are used to describe scene objects. Edges such as surface discontinuities are extracted using nonmaxima suppression on maxima of surface normals. Planar surfaces are then extracted by using a region-growing method. The system uses rigidity constraints to guide the matching process. At first, possible pairings between model and scene features are established, the transformation is estimated using quaternions. Then, further matches are predicted and verified by the rigidity constraints.

Ikeuchi: Ikeuchi [41] developed a method for object recognition in bin-picking tasks. Object models are generated under various viewer directions, apparent shapes are then classified into groups. The models consist of surface inertia, surface relationship, surface shape, edge relationship, extended Gaussian image, and surface characteristic distribution. Since this system is mainly designed for the task of bin-picking, only *one* type of object, which is the same one as in the model, appears in the scene. The same surface features used in models are extracted and classified by the help of the model. An interpretation is generated according to various model views. The orientation and location of the scene object are then decided by comparing their surface features and classified by the interpretation tree.

Fan et al. : Fan [26, 27] uses elementary surface patches as the primitives of description and recognition. The method consists of two stages. In the first stage, surfaces of 3D objects are segmented at discontinuities. Then these detected discontinuities are used to segment a complex surface into simpler meaningful components called surface patches. These patches can then be approximated by simple surface models. Finally, these surface patches are grouped to refer to meaningful 3D objects, and attributed graphs are generated to describe these objects. In the second stage, the descriptions of objects are used to build multi-view models. These views are arranged such that most of the significant features of the model object are contained in at least one of these views. Finally, partially occluded objects in a given scene are described using the same descriptions and identified as one of the model views. The recognition process contains three modules: the screener, the graph matcher, and the analyzer. In the first module, to avoid search for every possible match between model views and scene objects, for every scene object, all the model views are ordered according to their similarities to the scene object. Then, in the second module, a graph matching procedure is used, starting from the first ordered model view, to find the best correspondences between this view and the scene object. Finally in the

third module, the chosen matches are refined according to their geometric similarities. They present results on a variety of complex objects in scenes with multiple objects occluding each other.

Radack and Badler: The work conceptually closest to our approach was done by Radack and Badler [68]. Their representation of a surface patch is based on a radial decomposition. They introduce a new surface representation called *distance profile*. These profiles are used for the matching process. This method reduces the matching of three dimensional surfaces to the matching of two dimensional curves. They use points with high curvature to position the centers of the distance profiles. In their paper they present results with artificial data.

3D-POLY: With 3D-POLY Chen and Kak [18] developed a system in which they present a novel approach of organizing the feature data for three dimensional objects. They present a data structure which they call *feature sphere*. The matching and verification step is based on comparing spatial relationships of special feature sets. They show very fast recognition results for cluttered scenes with several industrial objects.

BONSAI: In [30] Flynn and Jain present a system, called BONSAI, which identifies and localizes instances of predefined three-dimensional models in range images. The models are generated with a CAD system. Relational graphs are extracted automatically from CAD models and are compared with a relational graph constructed from range image segmentation and surface classification modules. Recognition is performed via constrained search of the interpretation tree, using unary and binary geometric constraints to prune the search space. They show results on real data with up to two objects in the scene. The data base consists of twenty CAD objects.

Stark and Bowyer: Stark and Bowyer [77] implemented a system that takes the boundary surface description of a three-dimensional object as its input and attempts to recognize whether the object belongs to the category “chair” and, if so, into which subcategory it falls. This is the only implemented system to explore the use of a purely function-based definition of an object category to recognize three-dimensional objects. No explicit geometric or structural model is used. The authors evaluated the system on a database of over 100 objects, and the results largely agree with human interpretation of the objects.

Summary: This brief survey is summarized in Table 3.1. Many systems were developed which can deal only with a restricted set of object shapes, such as polygonal

Table 3.1: 3D from $2\frac{1}{2}$ D, 3D

shapes, solids of revolution or generalized cylinders. In contrast, we believe that our proposed system TOSS (Three dimensional Object recognition based on Super Segments) is able to recognize rigid objects, whose shapes are not constrained by any simplifying assumptions. Our algorithm uses a combined representation, which captures information about both smooth patches and discontinuity lines.

3.3 Our Approach

In recent years, object recognition in 3D has been either performed based on boundary and edge information (see [6, 44]) or by using surface descriptions (see for example [6, 26, 29, 67]). This creates problems when edges are not well defined, or when the objects cannot be segmented into stable elementary patches.

We present a system which combines both approaches with the following strategy: “Use for the recognition task whatever information is available”: We extract edges corresponding to depth and orientation discontinuities, and use them as primitives. These are not sufficient, however, to represent smooth, free form surfaces. So we also compute differential properties, called *splashes*, in smooth areas. We describe both of these primitives in the following subsections.

3.3.1 The 3D Curve

For some objects, such as polyhedra, it is natural to use a representation based on edges. For this reason, we extract 3D curves likely to correspond to depth and orientation discontinuities. Edges are in general broken (see e.g. in figure 3.2 the surface discontinuities between the fuselage and the wing). We take such inherent limitations into account. Our effort is not to develop a system which can only deal with *perfect* edges. We want to use whatever data current state-of-the-art edge detectors can generate. When we get non-invariant edges (such as limbs, which are viewer-dependent) we treat them just like all other edges. When they are matched against scene edges, they might generate wrong hypotheses, which are then discarded in the verification step. The most stable edges leading to the best matches are the edges which correspond to discontinuities of depth and surface (such as the boundary of the wings in figure 3.2).

Our representation of a general 3D curve is based on a polygonal approximation. We do not rely on any specific feature detection algorithm and we do not explicitly handle distinguished points such as corners or inflection points. Curvature and torsion are the most important features of a general 3D curve. They are invariant with respect to rotation and translation. By using a polygonal approximation we lose most of the curvature and torsion information, but we approximate it by computing the “curvature” (κ_j) and “torsion” (τ_j) angles between consecutive line segments (see figure 3.1).

Obviously, the polygonal approximation for a curve is not unique. Therefore, for the purpose of robustness, we use *several* polygonal approximations with different line fitting tolerances. Since we want to handle occlusion, we do not expect to obtain

Figure 3.1: Three-Dimensional Super Segment of Cardinality 4

complete curves in our scenes, but only portions of them. On the other hand, individual segments are too local to be useful as matching primitives. Grouping a fixed number of adjacent segments provides us with our first basic features, the 3D super segments. The 3D super segment is an extension to the 2D super segment which we used for recognition of flat objects in chapter 2. In accordance to figure 3.1, 3D super segments are characterized by their cardinality (number of segments), curvature angles (between consecutive segments), and torsion angles (between consecutive binormals).

As mentioned before, we are mainly interested in the curvature and torsion information implicitly captured by the 3D super segment curvature and torsion angles. This is the reason why we use them to encode a 3D super segment. The curvature (κ_i) and torsion (τ_i) angles are defined in the following way:

$$\kappa_i = \arccos \frac{s_{i+1} \cdot s_i}{|s_{i+1}| |s_i|} \quad \text{and} \quad \tau_i = \arccos(b_i \cdot b_{i+1})$$

with the binormals

$$b_i = \frac{s_{i+1} \times s_i}{|s_{i+1}| |s_i|}$$

and s_i is the i^{th} segment of the 3D super segment. To encode a 3D super segment ss with cardinality n we use a simple encoding scheme. The list of the quantized curvature and torsion angles values is the code of the 3D super segment ss :

$$\text{Code}(ss) = (\text{Quant}(\kappa_1), \text{Quant}(\kappa_2), \dots, \text{Quant}(\kappa_{n-1}), \text{Quant}(\tau_1), \text{Quant}(\tau_2), \dots, \text{Quant}(\tau_{n-2}))$$

Figure 3.2: Generation of 3D Curves

All the encoded 3D super segments serve as keys into a table (the data base), where we record the corresponding 3D super segments as entries, as explained later. We now address the following issues:

Curve Extraction So far we have not discussed the curve extraction process (see figure 3.2). We compute the edges of the range image. We use the edge detection algorithm proposed by Saint-Marc *et al.* [26, 27, 72]. It detects surface and depth orientation discontinuities. These features are inferred by examining the zero-crossings and extremal values of the surface curvature. The method uses adaptive smoothing to smooth a range image, which preserves discontinuities and facilitates their detection. This is achieved by repeatedly convolving the image with a very small averaging filter whose weights are a function of the local gradient estimate. In order to extract curvature extrema and zero-crossings, instead of smoothing the original range image R , the original derivatives $P = \frac{\partial R}{\partial x}$ and $Q = \frac{\partial R}{\partial y}$ are computed first. Then the images P and Q are repeatedly smoothed. Finally, the curvature values are computed from the smoothed images P and Q . The curvature extrema and zero-crossings are extracted using hysteresis.

Polygonal Approximation Because a polygonal approximation with a fixed tolerance is in general not stable, we use multiple line fitting tolerances. A complete analysis is given later. The 3D super segments are built by grouping adjacent segments. Because we cannot assign a specific direction to a 3D super segment, we use both directions for our representation.

Choice of Cardinality Which cardinalities should we use to define the link length of the 3D super segments? To use a fixed cardinality is possible, but it reduces the flexibility of the matching process. But when we have long 3D super segment matches, why not use them? Therefore we compute *all* possible cardinalities: that means that an open curve which is approximated by eight linear segments is represented by two 3D super segments of cardinality 8 (one for each direction), four 3D super segments of cardinality 7, six 3D super segments of cardinality 6, and so on. Higher cardinalities of matched 3D super segments increase the probability of having a *good* match.

3.3.2 The Splash

Basic Idea

For some objects, however, such as smooth objects, it is impossible to use edges for the representation. Therefore we come up with a new representation based on small surface patches where we can compute differential properties in a reliable way.

Extending the super segment idea is not straightforward. The polygonal approximation of a curve has a property which is crucial, but which is not extendable to higher dimensions: the well defined order of the neighborhood of a linear segment. Every segment on a polygon has two adjacent neighbor segments. Based on this fact, super segments can be generated by grouping adjacent segments together. In surface approximations, however, this ordered neighborhood property does not exist. Polygonal or other segmentations of a surface (or volume) lead in general to patches which can have any number and order of neighbor patches. This is a reason why we decided not to go the path of a polygonal (or higher order) surface segmentation to obtain a representation for matching and recognition. What are the requirements that a representation for general three dimensional objects has to meet? We want the representation to be

1. translation invariant,
2. rotation invariant,
3. general, in that we do not have to make any assumptions about the shape of the object,
4. local enough, so that we can handle occlusion,
5. robust enough, so that we can handle noise.

(a) Milk Splash

(b) Splash

Figure 3.3: Splashes

In the following, we use **lower case** letters to describe vectors (\mathbf{n} , $\mathbf{p} \dots$), and **upper case** letters to describe coordinate frames (\mathbf{N} , $\mathbf{O} \dots$). The basic feature for representing a general surface patch is the *splash*. The name originates from the famous picture of Professor Edgerton (MIT), showing a milk drop falling into milk (see figure 3.3 (a)). This picture bears a resemblance to the normals in our basic feature. A splash is best described by figure 3.3 (b). At a given location \mathbf{p} we determine the surface normal \mathbf{n} . We call this normal the *reference normal* of a splash. A circular slice around \mathbf{n} with the geodesic radius ρ is computed. Starting at an arbitrary point on this surface circle, a surface normal is determined at every point on the circle. In practice we walk around the reference normal with a $\Delta\theta$ angle (typically $1^\circ \leq \Delta\theta \leq 15^\circ$) and obtain a set of sample points on the surface circle. The normal at the angle θ is called \mathbf{n}_θ . A *super splash* is composed of splashes with different surface radii ρ_i with $i \in \{1, \dots, m\}$, where m is the number of splashes in a super splash.

In other words, the splash is the representation of a surface patch by the Gaussian map in the vicinity of the center of the patch, mapping the tangent with respect to a geodesic distance ρ . One question that is often asked: why do we not use Gaussian curvature for the splash computation (which is invariant to rigid transformations)? Why do we use the tangent information? The answer is straightforward: the computation of curvature requires a higher order derivative than the tangent. This implies that the signal to noise ratio is lower for a curvature based representation than for

(a) Relationship between \mathbf{n} and \mathbf{n}_θ

(b) Definition of ϕ_θ and ψ_θ

Figure 3.4: \mathbf{n} and \mathbf{n}_θ

a tangent based scheme. Therefore from the practical viewpoint we prefer a more reliable representation for the purpose of efficient matching.

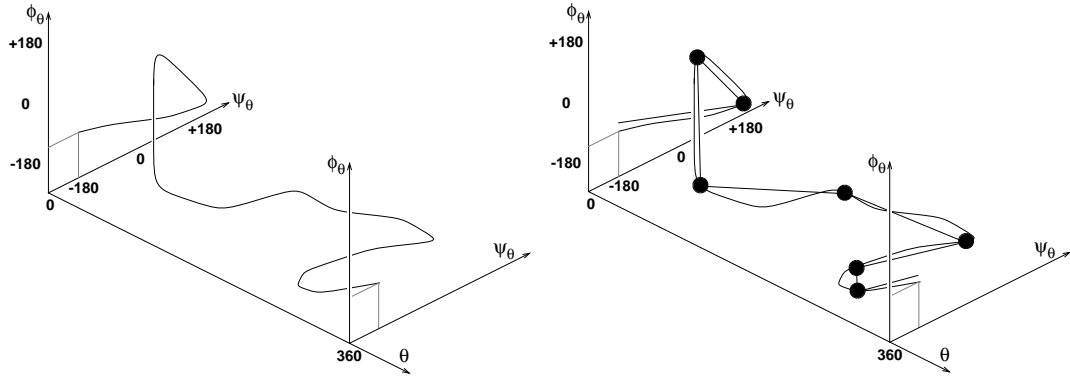
We compute a normal in our system by approximating the environment of a normal with triangular patches of small sizes. Every triangle votes for a triangle normal. The average of the three closest triangle normals is the surface normal. This is a very rough method, but the results were always good enough for our approach.

The frame \mathbf{N}_θ (see figure 3.4 (a)) is defined in the following way:

1. The surface normal \mathbf{n} is the z axis.
2. At every location of \mathbf{n}_θ , the location of the reference normal \mathbf{p} and the tip of the reference normal $\mathbf{n} + \mathbf{p}$ describe a plane E . The x axis is defined as the vector which is perpendicular to \mathbf{n} and lies in the plane E . Furthermore the angle between the x axis and a vector \mathbf{r} which is defined between the origin of Frame \mathbf{N}_θ and the location of \mathbf{n}_θ has to be in the interval $[-90^\circ, 90^\circ]$.
3. The y axis is perpendicular to the x and the z axis in a right handed coordinate system.

This frame has the property that the xy -plane always approximates the tangent plane of the surface in \mathbf{p} . We represent \mathbf{n}_θ in spherical coordinates: we compute the two angles ϕ_θ and ψ_θ :

$$\phi_\theta = \text{angle}(\mathbf{n}, \mathbf{n}_\theta) \quad \text{and} \quad \psi_\theta = \text{angle}(\mathbf{x}, \mathbf{n}_\theta^{z=0}).$$



(a) Mapping of ϕ and ψ into the (ϕ, ψ, θ) space (b) Polygonal Approximation of the Mapping

Figure 3.5: Vector Mapping $\vec{v}(\theta)$

For every sample point of a splash we obtain such a pair. Now we have a two dimensional mapping $\phi(\theta)$ and another one $\psi(\theta)$. In an earlier implementation (see [79]) we used these two mappings in parallel for our algorithm. But it is in fact possible to combine these two mappings into one compact three dimensional vector mapping $\vec{v}(\theta) = \begin{pmatrix} \phi(\theta) \\ \psi(\theta) \end{pmatrix}$, which describes a curve in the 3D space (ϕ, ψ, θ) . By doing so, we are able to use the representation for the general 3D Curve from section 3.3.1 for the representation of the mappings of the splash. Drawing a mapping for ϕ and ψ with respect to θ results in a mapping illustrated in figure 3.5(a). This mapping has the following properties:

1. Depending on where n_0 is, the mapping is shifted along the θ axis.
2. The mapping is periodic with respect to the θ axis.
3. The variation of the curve represents the structural change in the surface environment around the reference normal n .
 - (a) For a splash on a sphere or a plane, the mapping is constant.
 - (b) A creased surface results in a curved mapping.
4. Splashes which are located close to each other have a similar shaped mapping. By using the word *similar* we mean similarity in the sense, so that a human would classify them as “pretty much the same”. That does not automatically imply that the pairwise difference results in small values (we discuss the issue of robustness further in section 3.4). To be able to compare two mappings, we therefore need a difference measure, introduced below.

Encoding

At this point we have reduced the original question “How do we capture the shape of a general surface patch into a representation?”, into the much simpler question “How do we capture the shape of a mapping into a representation?”. The solution is straightforward based on our three dimensional approach for representing a general 3D curve (see section 3.3.1).

1. For all splashes of a model we compute their mappings. In section 3.3.2, we discuss the selection of the locations of the splashes.
2. For each splash, the mapping is approximated by polygonal approximations (see figure 3.5(b)). It is important to note that the mapping is periodic and therefore the polygon is closed. For the purpose of robustness we use multiple line fitting tolerances. Therefore we get a set of polygons for each mapping.
3. For every polygonal approximation we compute a 3D super segment. The start of the 3D super segment is defined at the point with the maximal distance of the θ axis. This corresponds to the point at which the sample normals have the strongest tilt with respect to the reference normal. If there is more than one global maximum we use one 3D super segment for each of the maxima. With this 3D super segment choice, we obtain rotational invariance in our representation. By starting all 3D super segments at the maximum of the approximation, two shifted polygons with the same shape produce the same 3D super segment.
4. All the obtained 3D super segments are encoded. The encoding works as described in section 3.3.1. As encodable attributes we take
 - (a) the curvature and the torsion angles of a 3D super segment,
 - (b) the maximum distance of the mapping from the θ axis,
 - (c) the surface radius of the splash.

Incorporated in the code of the angles of the 3D super segments is also the cardinality (number of segments) of the 3D super segments (by the number of angles). That avoids matching 3D super segments of different cardinality. The encoding of the maximal distance allows to distinguish between different curved surfaces of the same shape (e.g. two spherical surfaces with different sphere radii). The encoding of the radius avoids matches between splashes with different splash radii.

In summary, the code for one splash mapping is:

```
Code(splash-mapping) =
    (Quant( $\kappa_1$ ), Quant( $\kappa_2$ ), ...Quant( $\kappa_n$ ),
     Quant( $\tau_1$ ), Quant( $\tau_2$ ), ...Quant( $\tau_n$ ), Quant(max), Quant(radius))
```

where κ_i is the i^{th} curvature angle of the 3D super segment, τ_i is the i^{th} torsion angle of the 3D super segment, and n is the cardinality of the 3D super segment (n is implicitly encoded in the length of the code).

5. All the encoded 3D super segments serve as keys into a table (the data base), where we record the corresponding splashes as entries, as explained later.

Interest Operator

One question remains open: at which locations of an object should we compute the splashes? The brute force answer would be: at every pixel (in a range image). A more realistic answer would include the observation that we will not get *structurally rich* splashes at every point, which lead to good and unambiguous matches. Splashes in flat areas result in 3D super segments with extremely low cardinality (e.g. a splash on a plane maps on a 3D super segment consisting of one segment which corresponds to a cardinality of one). Super segments with such low cardinalities are less descriptive than super segments with higher cardinalities, which represent high structured surface patches. Therefore to obtain good and unique matches we are interested in matches of structured patches and high cardinality. These can be found at or near points of high curvature. Our simple selection method works as follows (see figure 3.6):

1. To compute the edges (surface and depth discontinuities) we use the algorithm mentioned in section 3.3.1.
2. We want to position the splashes in areas where we can expect structured patches on *one* object. This property is not given on a boundary. A boundary edge typically has the object as one neighborhood and other objects or background information as the other neighborhood. Therefore we use only the “inner object edges” and throw away the boundary edges.
3. For positioning the splashes we are interested in areas around areas of high curvature.

Placing a splash on a high curvature point has the disadvantage of an unreliable reference normal. A reliable reference normal is important for a stable splash. Nevertheless we want to capture the structure of the edges in the

splash. Therefore the best place for a splash is in the neighborhood of an edge. We get this area in three steps:

- (a) We dilate the edge image by replacing every pixel on the edges by a disc of a certain radius (e.g. $r_1 = 8$ pixels). The resulting image is called *dilatation 1*.
- (b) We dilate the edge image with another radius (e.g. $r_2 = 3$ pixels with $r_1 > r_2$). The resulting image is called *dilatation 2*.
- (c) The subtraction of dilatation 1 and dilatation 2 gives us a mask. This mask describes an area with the above described characteristics. Points in this mask are no high curvature points, but they are close to edges.

4. We compute a grid of splashes on the range image with respect to this mask.

This is obviously not the only way to choose the locations of splashes, but as we will see in the result section, this simple method works quite well. Therefore we have not emphasized this research direction.

(a) Scheme

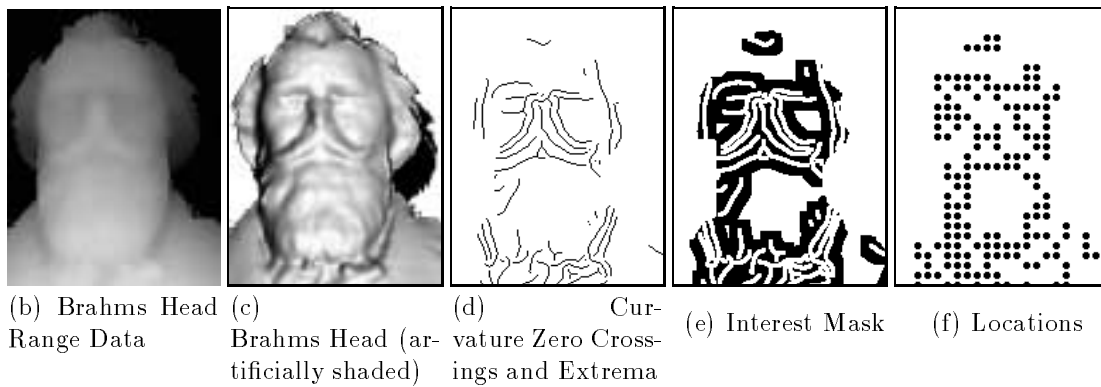


Figure 3.6: Interest Operator Scheme on an Example

3.4 Robustness and Stability of the Splash Representation

In order for our representation to be useful in recognizing objects from real data, it is necessary to address the issues of robustness and stability of the splash representation in the presence of noise.

location We cannot make the assumption that two splashes on different views of the same object are at *exactly* the same location with respect to the object. Therefore we have to examine the robustness of a splash regarding location uncertainty. We discuss the problem in section 3.4.1.

orientation Even when the location is correct, we cannot assume that the reference normal in both splashes is exactly the same. What are the effects on representation and matching of an error in the reference normal orientation? We address the issue in section 3.4.2.

stability How much noise can be added to the surface patch and still have a stable representation? We discuss the stability problem in section 3.4.3.

We have found empirically that our representation is adequate since our system performs well on real data. We would have liked to model the behavior of our scheme on arbitrary free form surfaces, but they are too general to be of any help. Instead, we present a full analysis on a simple analytic model of a specific surface patch, as shown in figure 3.7(a) and (b), for all three issues (location, orientation, and surface stability). Furthermore for the orientation robustness issue, we study the effects of noise in the general case of a 3D curve (representing either one of our features).

3.4.1 Robustness in the Location

To answer the question of how robust the representation of a splash is with respect to location accuracy, we use empirical data based on an example environment as described above. This range image was generated artificially and smoothed with a Gaussian filter. As shown in figure 3.7(a) our reference splash is centered south east of the corner. We show only the location of the splash. The corner has a height of 5 pixels. The reference splash has a radius of 30 pixels and the linear approximation of the mapping leads to a cardinality of 3. This result was observed for the whole set of fitting tolerances (15, 20, 25, 30). This means that the key consists of 6 values, 3 curvature and 3 torsion values. For reasons of simplicity, we did not encode the maximal distance from the θ axis. In our test results we got qualitatively similar

results as the ones shown here. We can also ignore the encoding of the splash radius, because in our example we deal only with splashes of one fixed radius. A qualitative side view of the reference splash, its sample normals and the shaded underlying surface is displayed in figure 3.7(b). The lines on the surface are artifacts from the rotation.

We use our reference splash and match it against a grid of 625 splashes (see figure 3.7(c)) in the vicinity of the reference splash spaced 2 pixels apart. In figure 3.8 we show the results for different quantizations of the matching key. We show in figure 3.8(a) the matched splashes between the reference splash in figure 3.7(a) and the splashes in figure 3.7(c), quantized with an interval size of 10° (the same result was observed for 20°). figure 3.8(b) shows the matches for a quantization of 30° , figure 3.8(c) for 40° , and figure 3.8(d) for 60° .

For small quantizations, the patch is recognized only if the splash is very close to the location of the original splash, whereas larger quantizations allow a much larger latitude in location uncertainty. Of course, if the quantization is too large (as in figure 3.8(d)), the splash may be confused with splashes from other locations. We therefore observe the desired robustness with respect to location uncertainty for this surface patch for the quantizations 30 and 40.

3.4.2 Robustness in the Reference Normal

Influence of Noise on the Mapping In this section we examine the influence of noise in the reference normal on the representation of a splash. In a first step, we focus on the effect on the mapping. In other words: adding an error angle ϵ in the

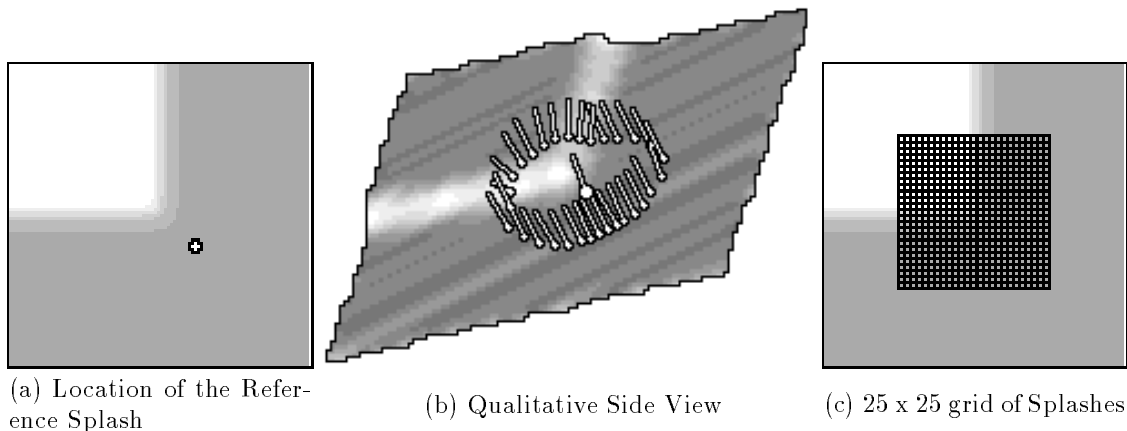


Figure 3.7: The Corner Surface

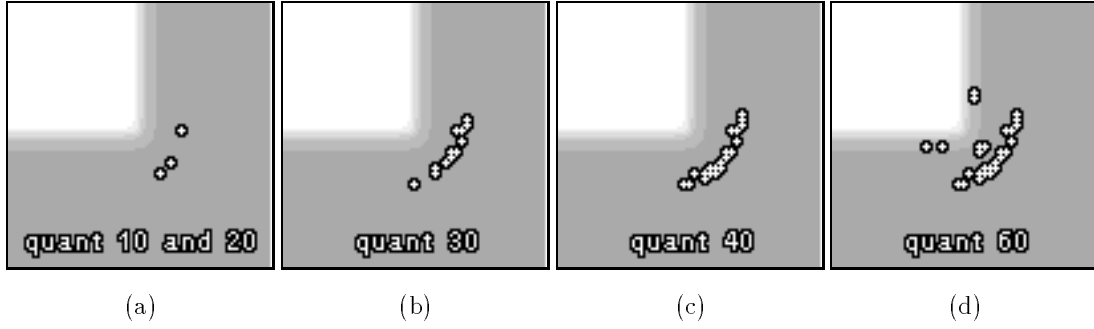


Figure 3.8: Matched Splashes with Different Quantizations

direction δ to the reference normal, how is the curve $\vec{v}(\theta) = \begin{pmatrix} \phi(\theta) \\ \psi(\theta) \end{pmatrix}$ influenced? In figure 3.11 we show the frame F which is the frame for a certain \mathbf{n}_θ . The frame F consists of the axis x , y , and z . As in section 3.3.2, z is equivalent to the reference normal \mathbf{n} . By adding an error angle ε in the direction δ to z we get z' . Without loss of generality we can set $\delta = \theta$. The frame F' consists of the axis x' , y' , and z' with

$$z' = \begin{pmatrix} \sin(\varepsilon) \cos(\theta) \\ \sin(\varepsilon) \sin(\theta) \\ \cos(\varepsilon) \end{pmatrix} \quad y' = z' \times \mathbf{r} \quad x' = y' \times z' = (z' \times \mathbf{r}) \times z'$$

with \mathbf{r} defined as in figure 3.4 in section 3.3.2. The computation of the change of ϕ with respect to θ is straightforward:

$$\begin{aligned} \Delta\phi_\theta &= \phi_\theta - \phi'_\theta \\ &= \arccos(z \cdot \mathbf{n}_\theta) - \arccos(z' \cdot \mathbf{n}_\theta) \\ &= \arccos(\mathbf{n}_\theta^z) - \arccos[\sin(\varepsilon) \cos(\theta) \mathbf{n}_\theta^x + \sin(\varepsilon) \sin(\theta) \mathbf{n}_\theta^y + \cos(\varepsilon) \mathbf{n}_\theta^z] \end{aligned}$$

The computation of $\Delta\psi_\theta$ is not that straightforward. The problem lies in the fact that the x axis is dependent on the data in the vicinity of the splash, expressed by the vector \mathbf{r} . To overcome this problem, we have to introduce some simplifying assumptions:

1. We can assume that ε is small (smaller than 10 degrees).
2. We can further assume that the angle between \mathbf{r} and x is small. We observed this angle for several thousand splashes on real data and we obtained the result that in approximately 75% of the cases the angle was smaller than 15 degrees. In figure 3.9 we show the histogram of the distribution of the angles computed

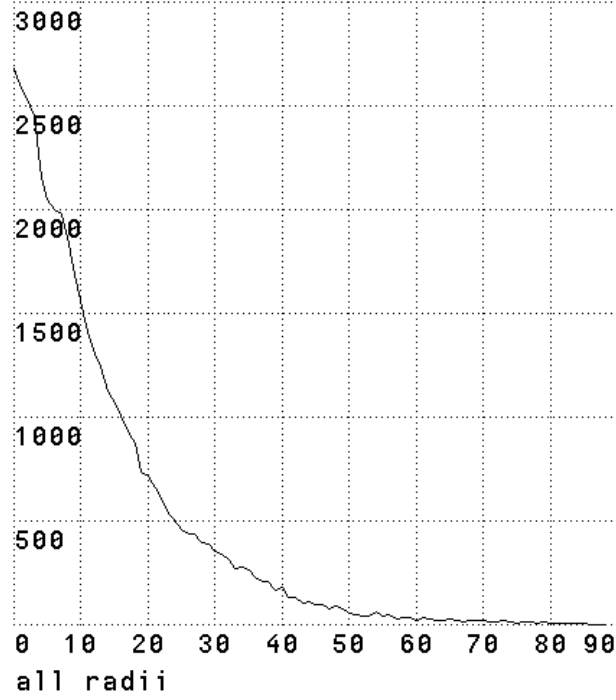


Figure 3.9: Histogram of Angle Distribution

from real data (we used the data of the busts, which we discuss in the results section). The abscissa represents the angles in degrees, the ordinate represents of the number of occurrences.

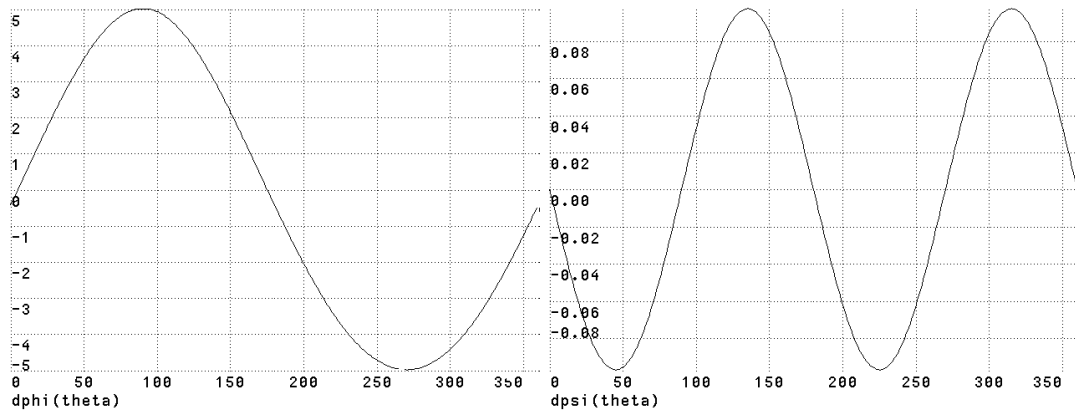
This allows us to approximate $r \approx x$ and $\mathbf{n}_\theta^{z'=0} \approx \mathbf{n}_\theta^{z=0}$. We therefore get:

$$x' \approx (z' \times \mathbf{r}) \times z'$$

and

$$\begin{aligned} \Delta\psi_\theta &= \psi_\theta - \psi'_\theta \\ &= \arccos(x\mathbf{n}_\theta^{z=0}) - \arccos(z'\mathbf{n}_\theta^{z'=0}) \\ &= \arccos(\mathbf{n}_\theta^x) - \arccos \left[\mathbf{n}_\theta^x \cos^2(\varepsilon) + \mathbf{n}_\theta^x \sin^2(\varepsilon) \sin^2(\theta) - \mathbf{n}_\theta^x \sin^2(\varepsilon) \sin(\theta) \cos(\theta) \right] \end{aligned}$$

The graphs of $\Delta\phi_\theta$ and $\Delta\psi_\theta$ with $\mathbf{n}_\theta = (0, \frac{1}{\sqrt{3}}, \frac{2}{\sqrt{3}})$ (assuming the sample normals are all equal) and $\varepsilon = 5^\circ$ are displayed in figure 3.10(a) and (b) respectively. We have verified qualitatively the shape of the curves with real data.



(a) $\Delta\phi(\theta)$

(b) $\Delta\psi(\theta)$

Figure 3.10: Example for $\Delta\phi(\theta)$ and $\Delta\psi(\theta)$

Our second step is to look at the maximum error on the curve $\vec{v}(\theta)$. Based on the triangle equation, which is also valid for the spherical geometry, we can bound the maximal error for $\Delta\phi$ by:

$$\|\Delta\phi\| = \|\phi - \phi'\| \leq \varepsilon$$

For the maximum of $\Delta\psi$ we can show under the above defined assumptions that the transformation T with $T : F \rightarrow F'$ is a rotation by the angle ε around an axis k which lies in the xy -plane and intersects the origin of F (and F'). This implies that the angle between x and x' can be at most ε . Therefore we get, based on the triangle equation, a bound for the maximal error for $\Delta\psi$ as

$$\|\Delta\psi\| = \|\psi - \psi'\| \leq \varepsilon.$$

We can now address the problem of how the representation might vary. Instead of a single curve $\vec{v}(\theta)$ we have now to deal with an envelope of curves $\vec{V}(\theta, \varepsilon)$ which is a disc of radius ε swept along \vec{v} (see figure 3.11(b)).

Influence of Noise on the Polygonal Approximation What are the polygonal approximations that approximate such a set of curves? How are the angles between consecutive line segments affected? To make a general statement is very difficult. Therefore we try to approach the problem by making some simplifying assumptions:

- We look at the two dimensional problem.
- We assume constant curvature.

(b) Envelope of Curve in Figure 3.5(a)

Figure 3.11: Uncertainty of the Reference Normal

(b) Closeup

Figure 3.12: Linear Approximation

That means, that we try to do a polygonal approximation along a circular section with a radius r_1 and we examine the changes of the polygonal approximation when we change the radius by an error ε to r_2 ($r_2 = r_1 + \varepsilon$, curvature $\kappa_i = 1/r_i$). In figure 3.12 we start the polygonal approximation with the line fitting tolerance t at the point P . The first segment from P to Q_1 is a linear approximation for the circle C_1 with the radius r_1 for the curve from P to Q_1 . The second segment from Q_1 to R_1 is a linear approximation for the circle C_1 for the curve from Q_1 to R_1 . The two segments form the angle α_1 . Changing the radius of the circle from r_1 to r_2 and starting the polygonal approximation at P with the same line fitting tolerance t leads to the points Q_2 and R_2 . The two segments form the angle α_2 . Our interest lies in the question of the difference $\Delta\alpha$ between α_1 and α_2 .

From figure 3.12, and with the relationship $t = r_i(1 - \cos(\mu_i/2))$ we can derive geometrically the following relationship between $\Delta\alpha = \|\alpha_1 - \alpha_2\|$, r_1 , r_2 , and the tolerance t :

$$\Delta\alpha = \left\| 2a \cos \left(\left(1 - \frac{t}{r_1}\right)\left(1 - \frac{t}{r_2}\right) + \sqrt{\frac{t^2}{r_1 r_2} \left(2 - \frac{t}{r_1}\right)\left(2 - \frac{t}{r_2}\right)} \right) \right\|$$

This equation is plotted in figure 3.13 for the values $\varepsilon = 5$ and $t = 5, 10, 15, 20, 25, 30$ with respect to radius r_1 . Our goal is to obtain a stable polygonal approximation. We choose a tolerance $t \gg \varepsilon$ for the following reasons (the numbers after the item correspond to the graph in figure 3.13):

Small Radius (0-20) A small radius, which corresponds to a sharp corner is a “wanted” breakpoint for the approximation. A small radius, which corresponds to noise, is ignored by an approximation with large t (see the start of the curves for the tolerances $t = 15, 20, \dots$ in figure 3.13). A small tolerance leads to extremely large values for $\Delta\alpha$.

Large Radius (50+) The approximation of parts of the curve with a large radius are very stable (see the low values for $\Delta\alpha$ in figure 3.13). The choice of the tolerance is not crucial. All tolerances provide small values for $\Delta\alpha$.

Medium Radius (20-50) The problems of an instable approximation start when the radius falls between the above discussed large and small radius. We observe two effects:

1. The values for $\Delta\alpha$ are fairly large.
2. The linear approximation might “break” at different points, dependent on little changes of the curve.

This is the reason why we use not one fixed tolerance for the approximation but a set of tolerances.

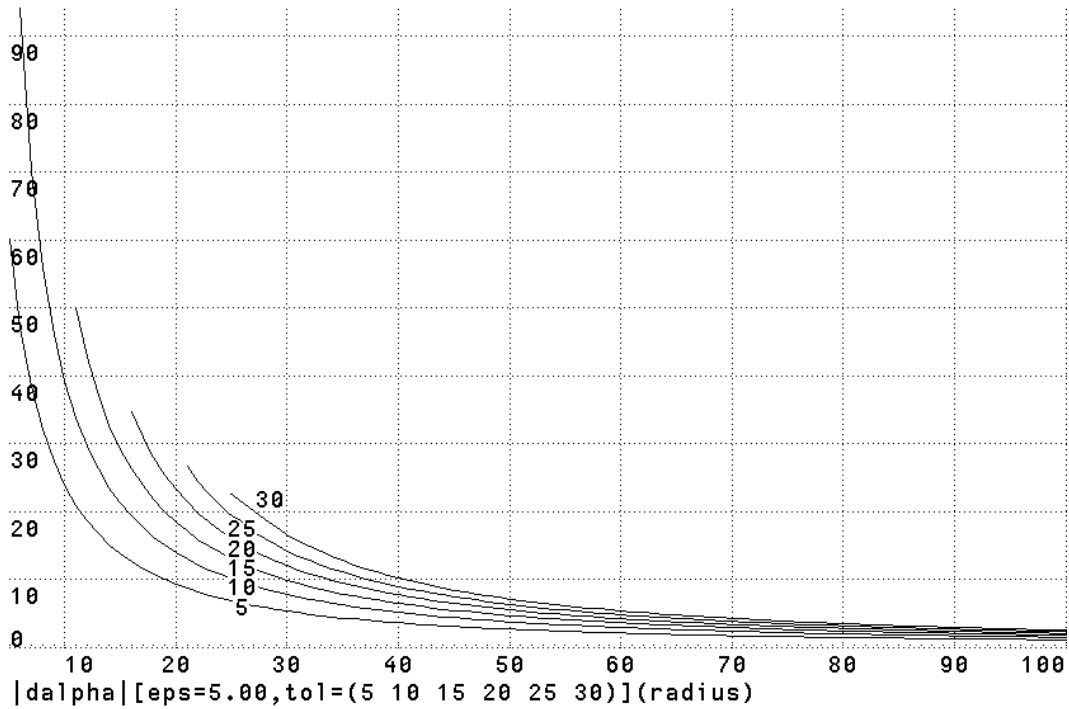


Figure 3.13: $\Delta\alpha$

We use the splash described in section 3.4.1 to examine its redundancy with respect to the reference normal. The results can be seen in the graphs in figure 3.14 for different quantizations (10, 20, 30, 40, and 60). The distance from the origin of the graph represents the angle α , and the direction, starting north and counting counterclockwise, represents the angle δ . Whenever a splash with the corrupted normal defined by (α, δ) matches the original splash (defined by the error $(0, 0)$), we plot a dot. Notice the different scales of the graphs.

To summarize, for small quantizations, the patch is recognized only if the reference normal is very similar to the reference normal of the original splash (see figure 3.14(a)). An uncertainty of up to two degrees is bearable and only for a certain δ direction. Larger quantizations allow a much larger error in the reference normal, and more freedom in the δ direction. The shape of the different distributions with respect to δ is dependent on the underlying surface, and the quantization.

3.4.3 Stability with respect to Noise

We want to address the question of how corrupted can the underlying surface be to reduce the stability of the splash representation. We added zero mean Gaussian

(b) Envelope of Different Quantizations

Figure 3.15: Stability Graph

noise $a\mathcal{G}(0, \sigma)$ with a the amplitude (in pixels) and σ the standard deviation, to the surface s to get the corrupted surface cs :

$$cs(x, y) = s(x, y) + a\mathcal{G}(0, \sigma)$$

Then we compute the splash at the same location as the reference splash on the corrupted surface. For a given quantization we then match the “corrupted” splash against the reference splash. If both splashes match, we put a point in the graph for the corresponding quantization at the coordinate (σ, a) . We show the scatter graph for the quantization 40 in figure 3.15(a) and the graphs of the envelopes for the quantizations 10, 20, 30, 40, and 60 are shown in figure 3.15(b) respectively.

Our conclusion is that the larger the quantization, the more stable is the splash representation, the smaller the quantization, the less stability we can observe. The matching behavior is not affected by either increasing the amplitude of the noise and keeping the standard deviation small or decreasing the amplitude and having a large noise deviation.

3.5 Recognition

3.5.1 Object Representation

As mentioned in the previous sections, we want to represent our model (or scene) with super segments for curve representation and splashes for surface patch representation. We want the representation to be compact, fast accessible, and the storage of multiple objects should be possible. We chose for these reasons a table which is implemented as a hash table (more about hashing see [96]). A hash table allows efficient storage (only pointers are recorded), the indexing scheme allows fast access and different features with the same keys can be stored in cellar like buckets.

The representation of an object consists of the following steps (see figure 3.16):

1. Compute the features mF_i of model \mathbf{m} with respect to the algorithms described in section 3.3.1 for super segments and section 3.3.2 for splashes.
2. Encode the features:
 - Encode the curvature and torsion angles for the 3D super segments (section 3.3.1).
 - Encode the curvature angles and the other attributes of the mappings of the splashes (section 3.3.2).

Figure 3.16: Object Representation

3. For every feature mF_i several codes $\text{Code}_j({}^mF_i)$ are computed. They are related to the different line fitting tolerances. Every code of every feature serves as a key for an entry in a table (the data base) where we record the feature.

When we build the data base for more than one object we perform the three steps for every object. The table (data base) grows in size with the number of recorded models. This process of building the data base can be done off line.

3.5.2 Hypotheses Generation

Candidate Retrieval: The task of hypotheses generation is the process to establish correspondences between features of stored models and features of the scene. This process results in a set of matching hypotheses which consist of *good* and *false* matches. To separate the good hypotheses from all hypotheses we have to find consistent clusters; this process is discussed in section 3.5.3. Our main interest for the candidate retrieval lies in the discriminative power of the hypotheses generation itself. By using indexing, we gain a lot of this power. Several systems of the past (see section 3.1) use in this respect very weak features like points or edges. This leaves all the discriminative work to the verifying step. We believe that our indexing mechanism reduces the ratio of false hypotheses to good hypotheses tremendously. One positive side effect is that, in our experiments, models which were not in the scene and therefore provided only false matches had very few hypotheses and could be excluded fairly fast by the verification.

The hypotheses generation consists of the following steps:

Figure 3.17: Hypotheses Generation

1. The scene is preprocessed to generate all the features F_k (splashes and super segments) as explained in section 3.5.1.
2. The features are encoded.
3. The encoded features are used to retrieve the candidate hypotheses between the features of model and scene.

Quantization and Cardinality: We discussed in section 2.5 the crucial encoding parameter of the quantization size. The same truth holds for the three-dimensional features. Two features match, when both keys are exactly the same. This means that all the pairwise values have to fall into the same quantization intervals.

Looking at equation 2.1 in section 2.5, it is obvious that the same quantization for different keys increases the probability for matches of small cardinality and decreases the probability for matches of large cardinality. In our implementation we try to counteract this effect by using larger interval sizes for larger cardinalities. Typical values are (only for the quantization of the curvature and torsion angles):

cardinality	3	4	5	6	7	8	...
number of keys	6	8	10	12	14	16	...
interval size	30	40	45	60	60	90	90

For extremely noisy data we use slightly larger values, for data with little noise we use smaller values.

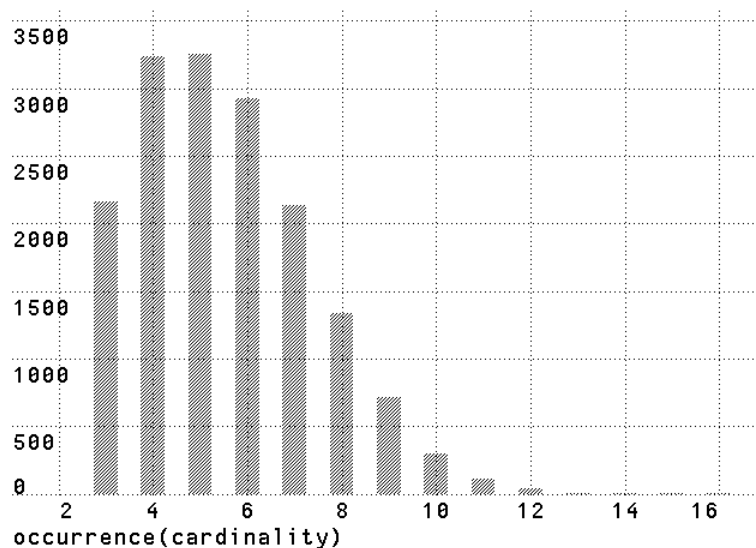


Figure 3.18: Occurrence Distribution with respect to Cardinality

What kind of splashes do occur in typical range data? Statistics for the three composers (see the results section) and the combined scene gives us the histogram in figure 3.18. From our experience we get the best matches from the cardinalities 3 to 7. We believe that this is mainly due to the lower probability of matches for larger cardinalities in the equation 2.1 which we discussed in section 2.5.

3.5.3 Verification

The task of the verification is to distinguish *good* from *bad* hypotheses. Good hypotheses correspond to true matches, bad hypotheses correspond to wrong matches. Good hypotheses have the following properties:

- they correspond to a rigid transformation,
- they can be grouped in geometrically consistent clusters.

Therefore, the verification stage consists of the following steps:

1. We compute all possible matches for the features of the scene with the model features to generate multiple hypotheses. We remove the hypotheses which do not represent a rigid transformation. This can be done for every hypothesis by computing a least squares match between the model feature and the scene feature. The determinant of the resulting rotation matrix should be approximately 1.0 to represent a rigid transformation (see Appendix A). Next, we

Figure 3.19: Verification

divide the resulting n hypotheses $H = \{h_1, h_2, \dots, h_n\}$ according to which model the model feature of the hypothesis votes for. We store these into a correspondence table where we have the models m_i as keys and the i_k hypotheses $H_i = \{^i h_1, ^i h_2, \dots, ^i h_{i_k}\}$ (with $H_i \subseteq H$) as entries (see figure 3.19).

2. The next step is the formation of consistent clusters. For every model m_i we have to check which hypotheses $^i h_x$ and $^i h_y$ with $^i h_x \neq ^i h_y$ are consistent with each other. In theory one matched feature is enough to establish the complete transformation between model and scene. In practice we have to consider several aspects:

- The splash is very local.
- Due to this locality, noise has a lot of influence on the transformation.

Therefore we view, for the geometric analysis, a hypothesis as a match equivalent to a point match. In practice we use the locations of the splashes as the matching point pair. And because three non-collinear point matches define a unique rigid transformation in three dimensions, we adopt the criterion that three consistent hypotheses are sufficient to instantiate a model in the scene. We do not check every hypothesis against every other, instead, if we have three consistent hypotheses $C = \{^i h_r, ^i h_s, ^i h_t\}$ with $C \subseteq H_i$ for one model m_i , we

examine the remaining hypotheses in $H_i \setminus C$ and collect those that are consistent with at least one of the selected three in C . When we have found one instance, represented by $I = C \cup F$, with F the additional found consistent hypotheses, we try to find more instances in the remaining hypotheses $H_i \setminus I$. But what is meant by consistency? We use the powerful geometrical constraints (distance, direction and angle) introduced by Grimson and Lozano-Pérez [34, 35] to prune efficiently their interpretation trees, and build our clusters. For more detailed discussion of the geometrical constraints, see Appendix B. In the three dimensional domain, these three constraints define the attitude of one feature relative to another since it specifies the five degrees of freedom (three translational for the position and two rotational for the orientation).

3. After this grouping of hypotheses into clusters, we can compute the transformation from the model coordinates to the scene coordinates by applying a least squares calculation on all the matching features (see Appendix A). Because of noise, we get in general a good first guess for the transformation but not an exact match. A second least squares match on corresponding corners or segments can refine the result.

3.6 Complexity Analysis

The whole complexity analysis focuses on the question: “What is the discriminative power of the features in the system?” The answer to this question can be split in half. The first part is the analysis of the retrieval process and the number of generated hypotheses. The second part is the discussion of the verification step and the cost of grouping them into consistent clusters. When we talk in the following about “scene features” we consider only the scene features, which lead to the generation of the minimum of one hypothesis. For the complexity analysis, we ignore the scene features whose code is disjunct from all the keys in the table (they also do not add any cost in practice). Before we start to answer the above question we look at the parameters involved:

- n = the number of features in the scene,
- d = the number of features per table entry,
- m_s = the number of models in the scene,
- and m = the number of models in the data base.

To simplify the discussion, we make the assumptions that every model has the same number of super segments, and that the entries are equally distributed over the table ($d = \text{const}$). Furthermore we assume that every model consists of f features ($f = \text{const}$). The analysis is valid for the TOSS system and the two-dimensional system described in chapter 2.

3.6.1 Hypotheses Generation

When we have n features in the scene, it is obvious that the cost to generate all candidate hypotheses with indexing is $O(n)$. The important issue for the cost of the complete recognition is *not* the cost of retrieval, but h , the number of hypotheses generated, because this number has a crucial effect on the verification step. The number h is proportional to the number of features per table entry d ($h = d \cdot n$). The larger h , the slower is the clustering into consistent clusters in the following verification. Therefore we are interested in a small h , which corresponds to a small d . This, so called “ability to discriminate”, is influenced by the following factors:

Noisy Data: Noisy data in the scene and in the models forces the use of larger quantization intervals for the encoding of the features. Larger quantization increases the number of retrieved hypotheses, and therefore decreases the ratio of good versus bad hypotheses in the retrieval phase (see section 3.5.2).

Similar Models: Similar models consist of similar features. Obviously, similar features are less discriminative than distinct features, and therefore d , the number of features per table entry increases. As a result, a scene feature generates an increased number of candidate hypotheses. The work load of clustering can only be done based on geometrical constraints and is left to the verification step.

Grid Size: Choosing a large grid size for the interest operator, we require the system to be robust to a larger location uncertainty. This forces the user to choose a larger quantization for the encoding of the features and therefore results in a smaller ratio of good versus bad hypotheses.

To summarize, the crucial value for the retrieval stage is h , the number of generated hypotheses. In a best case, this number is very low. The theoretical best case is, when every hypothesis votes for a different model ($d = 1$ and $n = m_s$). This results in $h = n$. The worst case corresponds to a large value of h . There is only little discriminative power in the features. This means that most features are encoded with the same code (large d). Every scene feature generates all possible candidate hypotheses and therefore the overall number of retrieved hypotheses candidates is

$h = f \cdot m \cdot n$. This is the approach taken by many systems of the past, which use either points or lines as basic features, and must therefore perform the discrimination task during the verification step.

3.6.2 Verification

As mentioned above, the task of the verification is to cluster the h hypotheses into mutually consistent clusters. This is done for every entry in the correspondence table.

Best Case: In the best case, there is a lot of discriminative power in the features. This corresponds to a low d value. The $h = d \cdot n$ hypotheses are divided in the correspondence table according to which model the hypothesis votes for. For m_s models in the scene we have m_s entries in the correspondence table with h/m_s hypotheses each. For every entry these h/m_s hypotheses have to be grouped with respect to the geometrical constraints. We distinguish the clustering process based on different cases:

- *Every model in the scene occurs only once:*

In this case, the hypotheses in one entry either vote for the model instance or they are wrong hypotheses. The grouping of the hypotheses can be done by finding three consistent hypotheses (see section 3.5.3) and then screening the remaining hypotheses in the entry for consistency with the initial three. Under the assumption that the number of wrong hypotheses is relatively small, this grouping is done in $O(h/m_s)$. For the whole complexity we therefore get

$$O_{best}(n) = m_s \cdot O(h/m_s) = O(d \cdot n) = O(n).$$

The theoretical absolute best case is when $d = 1$, and every hypothesis votes for a different model ($n = m_s$). The correspondence table has m_s entries with one hypothesis each. The cost for the clustering is zero. This results in $O_{best}(n) = O(1)$. (Note that the pose estimation computed from a single feature is likely to be very coarse.)

- *Every model in the scene can occur more than once:*

In this case the clustering of the hypotheses in one entry cannot be done in $O(h/m_s)$. To find the three initial hypotheses for every instance, we get the complexity of $O(h^2/m_s^2)$. Clustering these hypotheses for all entries results in a complexity of

$$O_{best}(n) = m_s \cdot O(h^2/m_s^2) = O(d^2 \cdot n^2/m_s) = O(n^2).$$

The best case has a noteworthy side effect. Assuming the number of scene features n fixed, and examining the complexity O_m with respect to the models in the data base, we find that it grows as $O_m = O(k \cdot m)$ when m is the number of stored models, and $k < 1$. To show this effect, we performed two experiments:

1. Large data base using only edge information: We created a data base of 100 random polyhedra consisting of mutually intersecting random tetrahedra and parallelepipeds. The scene was composed by taking three of these polyhedra, rotating and overlapping them. For the recognition we used only 3D super segments. The cost for detecting the absence of a model is less than 10% of the cost for the detection of the occurrence. This underlines the discriminative power of super segments in three dimensions.
2. Large data base using only splashes: We created a data base of 100 random smooth range images. The scene is a composite of four of these objects including translation, rotation, and occlusion. For the recognition we used only splashes. In our results, the cost for detecting the absence of a model is less than 50% of the cost for the detection of the occurrence. We believe that the cause for the higher relative cost of absence detection based on splashes compared to the super segments lies in the fact that splashes for surfaces are less descriptive than super segments for boundaries.

The experiment gives us an upper and lower bound for what we have to expect for the detection of occurrence and absence of a general three dimensional object using a large data base in the case of discriminative features. Therefore the complexity with respect to the number of models stored is $O_m = O(k \cdot m)$ with $0.1 < k < 0.5$.

Worst Case: In the worst case, there is little discriminative power in the features. This corresponds to a high value for d . The overall number of retrieved hypotheses candidates is $h = d \cdot n$ with $d = m \cdot f$. These h hypotheses are divided in the correspondence table according to which model the hypothesis votes for. For m models in the data base we get, in the worst case, m entries in the correspondence table with h hypotheses each. For every entry, these h hypotheses have to be grouped with respect to the geometrical constraints. Clustering these h hypotheses results in a complexity of

$$O_{worst}(n, m) = m \cdot O(h^2) = O(f^2 \cdot n^2 \cdot m^3) = O(n^2 \cdot m^3).$$

In the worst case, the ratio of good versus bad hypotheses is very small.

3.6.3 Summary

As a conclusion, we get the result that the practical complexity of our system is

$$O(n) \leq O_{\text{recognition}} \leq O(n^2 \cdot m^3).$$

In the case of well distinguishable models, such as the first two examples in the results section 3.7, the complexity comes close to the above discussed *best case*. An example where the system slows down is shown in the third example of the results section 3.7, which presents the results of a cluttered scene, composed of three composer busts. The system detects the correct models and computes the correct locations, but due to noisy data and similar features, the discriminative power is smaller, and the overall recognition process is slower than in the other examples.

3.7 Results

The recognition mechanism for general three dimensional objects is now illustrated with real data examples. For the presentation of the range data we always display the artificially shaded images. We choose four scenes:

1. a Mozart bust, which is not segmentable into stable patches,
2. a plane and a wagon, which shows that our method works for objects which can be approximated by polygonal surfaces (this input was used by T. J. Fan in [27, 26]),
3. a very complex and cluttered scene with similar objects (busts of composers),
4. a terrain scene lacking significant features, which is an interesting application regarding navigation tasks.

The Mozart scene and the plane-and-wagon scene were recognized with a data base consisting of 9 objects. The contents of the data base is shown in figure 3.20.

3.7.1 Mozart

Our Mozart bust is highly curved, and has a partially structured surface. The range data was obtained with a laser range finder. Because of lack of data we cannot deal with a complete three dimensional model and a scene which consists of range

data. Therefore we take the original data of the Mozart bust as model, rotate the range data synthetically to obtain the scene. We rotate pixel by pixel and fill the holes by averaging the values of neighbor pixels. This rotation process is guaranteed to add a lot of noise! Our input data is the range image. For better visibility we show the artificially shaded images. Figure 3.21(a) shows the rotated scene of the Mozart bust, which is the model (see figure 3.20) rotated by 20 degrees around a tilted axis. The recognition result is shown in figure 3.21(b). (We overlaid a grid of the range image of the model, transformed by the resulting transformation on top of the figure 3.21(a).) In figure 3.21(c) and (d) we show the final hypotheses which lead to the result in figure 3.21(b). Hypotheses 2 and 3, and 18 and 19 are overlaid.

3.7.2 Plane and Wagon

We have four range images, two of the plane from different views and two of the wagon from different views. The range data of all four views was obtained with a laser range finder. One wagon and one plane image serve as models. The scene is composed synthetically by combining the other two range images into the scene image as displayed in figure 3.22(a). Figure 3.22(b) shows the best detected solution. In figure 3.23 we display the final hypotheses. The splash hypotheses are displayed *italic* and the super segment hypotheses are printed **bold**. Splash hypotheses are shown as the pointers to the corresponding splash location, and super segment pointers point to the middle vertex of the super segment (to display the super segments themselves is difficult). Whenever more than one feature pointer points to a location, we have multiple feature matches based on different linear approximations (for super segments) or different radii (for splashes). It is interesting to note that the plane was recognized based on the super segment information (only the hypothesis #2 in figure 3.23(a) and (b) is a splash hypothesis), whereas the wagon was mainly detected based on splashes (only the hypotheses #(1-3) in figure 3.23(c) and (d) are a super segment hypotheses). This is a good illustration of the system's ability to use whatever information is available: for the wagon the splashes are the best matched features, for the plane the 3D curves.

3.7.3 Composers

We obtained three range images from the three busts shown in figure 3.24(a-c) with a liquid crystal range finder [74]. These three busts serve as models and are the content of the data base. The scene is composed by overlaying three different views of the busts (the range finder's field of view and depth of field are too small to acquire such an amount of data at once). The data is smoothed with the adaptive smoothing

algorithm [72] and small holes are closed with bilinear interpolation. The scene is displayed in figure 3.24(d). The algorithm finds the correct correspondences and the correct positions despite the fact that the three models are locally very similar (parts of the faces, parts of the clothes). The hypotheses generation step retrieves all possible candidates and the verification step has to unravel the consistent clusters. The projection of the detected models on the scene is shown in figure 3.25. This example shows the limits of our algorithm with respect to speed. The detection is by far the slowest compared to all the other scenes (see the complexity discussion in section 3.6). This is mainly due to the similarity of the objects, the noisy data (the liquid crystal range finder provides fairly noisy data due to interreflections and coarse quantization), and the complexity of the scene. It is interesting to note that the algorithm found multiple solutions (e.g. an instance of the Chopin model was also found on Bach in the scene) but they were later rejected based on the rigidity assumption.

In figure 3.26 we show as an example the hypotheses which lead to the recognition of Chopin. Hypotheses 11 and 13 illustrate very well the location uncertainty.

3.7.4 Terrain Map

As a terrain map we use the DEM (digital elevation model) covering the Martin Marietta ALV test area. A digital elevation model is a two-dimensional array of uniformly spaced terrain elevation measurements. Our DEM map consists of 810 x 702 pixels with a pixel corresponding to a size of 5 x 5 meters on the ground. For better visibility we exaggerated the elevation data. The DEM map serves as the model in our data base. We cut out a window of 80 x 80 pixels, rotated it by 80° around a tilted axis and used the resulting range information as the scene data. This kind of recognition can be useful in navigation or mapping tasks. Figure 3.27 shows the shaded range image of the terrain map. The rectangle represents the area we picked for the window. Figure 3.28(a) and (b) show the range and the shaded image of the window, which was rotated by 80° around a tilted axis. The marked arrow in figure 3.27 shows from which direction we look at this window to get the view in figure 3.28. The final hypotheses are displayed in figure 3.28(c) and (d). Figure 3.29 shows the best match.

3.7.5 General Observations

We can give some rough numbers about the running time (on a serial Symbolics 3675 Lisp machine).

1. The acquisition of one super splash (consisting of typically 3 splashes with 4 line fitting tolerances) takes about 12 seconds.
2. The Mozart bust consists of about 400 super splashes, therefore it took about 1 hour 20 minutes to compute all splashes.
3. The plane consists of about 60 splashes, therefore it took about 12 minutes to compute all splashes.
4. The computation of the 3D curves takes less than 3 minutes.
5. The recognition process for the mozart scene takes about 50 seconds (retrieval: 10 seconds, verification: 40 seconds).
6. The recognition process for the plane and wagon scene takes about 1 minute 30 seconds (retrieval: 15 seconds, verification: 75 seconds).
7. The recognition process for the bust scene takes about 30 minutes (retrieval: 2 minutes, rigidity filter: 7 minutes, verification: 21 minutes).
8. For the window of the DEM map we compute 109 splashes (which takes approximately 20 minutes). The recognition itself succeeds after 35 seconds (retrieval: 5 seconds, verification: 30 seconds).

All these numbers reflect neither the high parallelism which is theoretically possible nor the data redundancy with which we work at the moment. Simple improvements can significantly increase the performance. This is one goal of our future work.

object (size in pixels)	# super splashes	# super segments	splash radii (pixels)	grid (pixels)	recognition time
car <small>(387x239)</small>	235	208	20,30,40	8	
dinosaur <small>(141x305)</small>	60	64	20,30,40	8	
mask <small>(294x290)</small>	350	276	20,30,40	8	
Mozart <small>(264x399)</small>	402	not computed	20,30,40	6	
phone <small>(395x217)</small>	262	193	20,30,40	8	
plane <small>(507x218)</small>	60	239	20,30,40	8	
pyramid <small>(105x97)</small>	27	not computed	20,30,40	5	
Renault part <small>(315x367)</small>	124	204	20,30,40	8	
wagon <small>(422x178)</small>	367	256	20,30,40	6	
Bach <small>(230x389)</small>	506	not computed	15,20,25	6	
Brahms <small>(232x351)</small>	640	not computed	15,20,25	5	
Chopin <small>(232x359)</small>	469	not computed	15,20,25	6	
scene 1 <small>(240x390)</small>	329	not computed	20,30,40	8	50 s
scene 2 <small>(458x324)</small>	293	350	20,30,40	8	90 s
scene 3 <small>(446x424)</small>	849	not computed	15,20,25	7	30 min
DEM window <small>(80x80)</small>	109	not computed	15,20,25	3	35 s
DEM map <small>(811x702)</small>	5544	not computed	15,20,25	5	

Table 3.2: Table of three-dimensional Objects

Figure 3.20: Data Base with 9 Objects

(a) Scene 1

(b) Detected Model Projected
on Scene 1

(c) Final Hypotheses of Mozart

(d) Final Hypotheses of Scene

Figure 3.21: Example Mozart

(a) Scene 2

(b) Detected Models Projected on Scene 2

Figure 3.22: Example Plane and Wagon

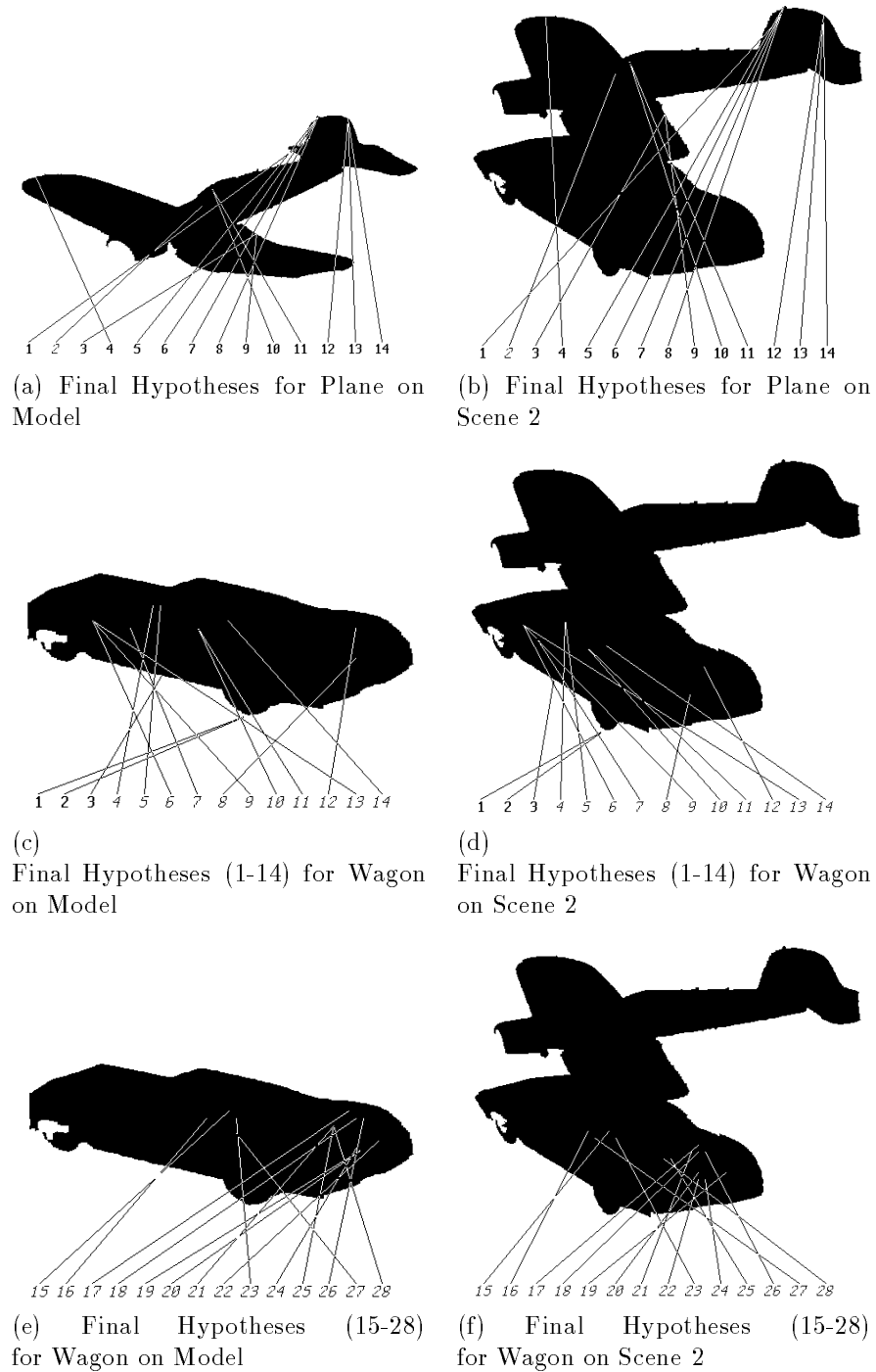


Figure 3.23: Hypotheses for Scene 2 (splash hypotheses are printed *italic*, super segment hypotheses are printed **bold**)



(a) Bach

(b) Brahms

(c) Chopin



(d) Scene 3: Three Composers

Figure 3.24: Three Composers

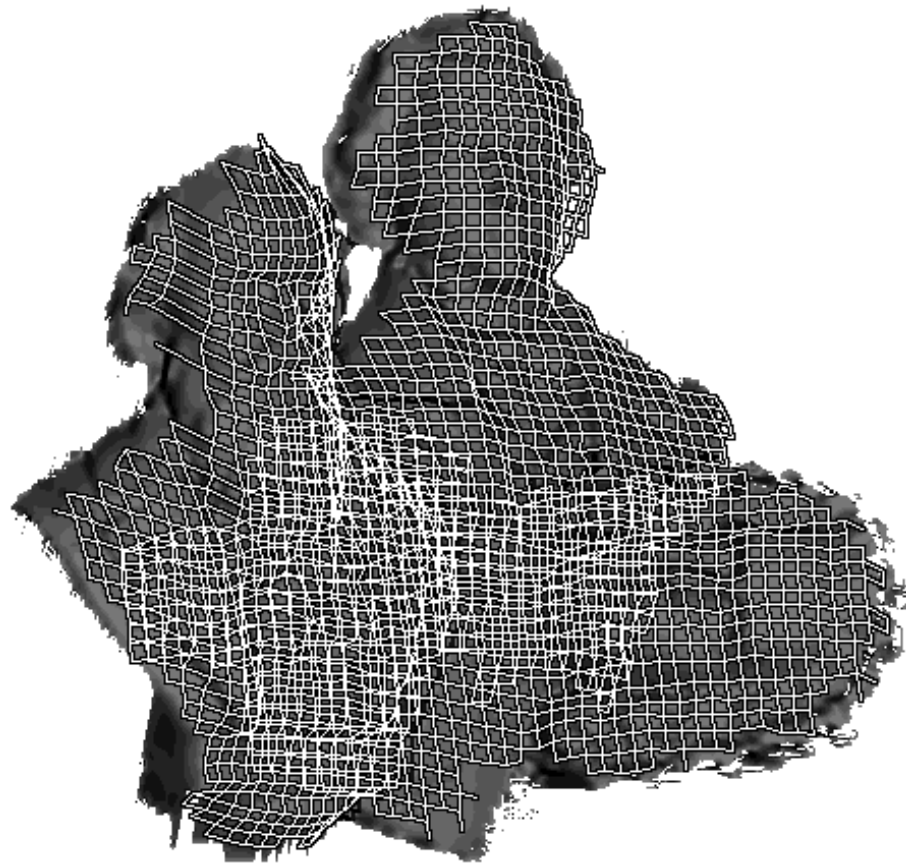
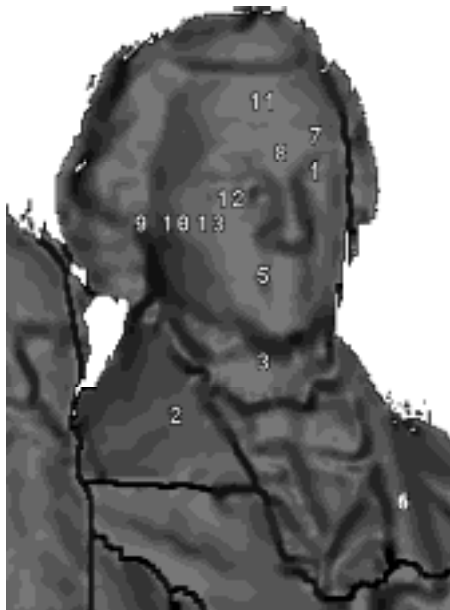


Figure 3.25: Projection of Detected Busts in Scene 3



(a) Hypotheses on Chopin



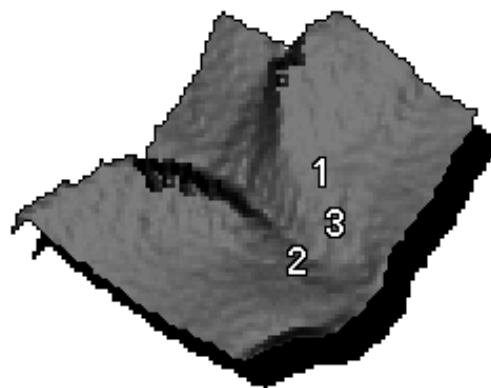
(b) Hypotheses on Scene 3

Figure 3.26: Example of the Hypotheses found for Chopin (Hypotheses 4 and 6 are overlaid in the Scene)

Figure 3.27: Shaded DEM Terrain Map (= Model)

(a) Range Image of Window

(b) Shaded Image of Window



(c) Final Hypotheses on Closeup View of Model

(d) Final Hypotheses on Scene

Figure 3.28: Rotated DEM Terrain Window (= Scene) and Final Hypotheses

Figure 3.29: Result of Recognition (Scene projected on Model)

3.8 Conclusion

We have shown with our implementation of the TOSS system that structural indexing provides a powerful mechanism for the recognition of general three dimensional objects. We make very few restrictive assumptions about the shape of the objects, and we are able to acquire them automatically. By using two types of primitives, we overcome the problem of recognition in the case where either edge data or surface data provides not enough information for a correct classification. Our encoding scheme allows us to match the primitives and verify the resulting hypotheses in a reasonable time complexity. We are able to handle large object data bases.

Future systems have to further exploit the fact that *rich* features such as the splash or the super segment provide enough structural information to recognize objects efficiently.

Chapter 4

Recognition of Three-Dimensional Objects from Two-Dimensional Images

4.1 Introduction

In this chapter we propose an approach for the recognition of three-dimensional objects in a two-dimensional scene. As models, we use a set of non-registered views of a three-dimensional object. By using perceptual organization we develop a hierarchy of features based on proximity, symmetry, parallelism, and closure. The detection of these features is performed in an efficient way using *proximity indexing*. While most other systems use spatial relations for the recognition process, we use high level features and their *topology*. Using indexing, we retrieve matching hypotheses, which are verified against each other with respect to topological constraints. Groups of consistent hypotheses represent detected model instances in a scene. We show some results with our current system.

Most object recognition systems today (including ours, see chapter 2 and 3) address the problem of finding the location and orientation of an exactly known rigid object in a scene. The tools used to achieve this task are *geometric constraints*, and a lucid treatment of this class of approaches can be found in Grimson's book [33]. The presence of a model is inferred by the verification that such a model could indeed produce some of the observed data under an appropriate geometric transform. It is clear that, with this approach, one can use local, low level primitives such as edgels or their approximations, as produced by state of the art edge finders. Such an approach is therefore very appropriate when evolving in a controlled environment, such as a factory, where the number of possible objects is small and their geometry is precisely known, but cannot be extended to more general scenarii for the following two reasons:

- It is quite difficult to build accurate geometric models (unless they are designed that way).

- When the object library is large, it becomes necessary to develop methods for indexing into the library to select likely objects.

One possible way to circumvent these problems is to use projective invariants [31], but these lack the discriminative power which we need to distinguish several models. Furthermore they might be difficult to detect and are sensitive to occlusion.

We propose instead that the solution to these problems lie in *grouping* of the initial primitives. Such groups serve as an intermediate level representation of the data, in a hierarchical fashion, and can be used to retrieve likely candidate objects from a library. Furthermore, these can be extracted from multiple views of an object, rather than from a complete model (as such they are “quasi-invariants” [8, 54]). This idea of “perceptual grouping” is hardly new. It can be found in the psychology literature of the 1920s, under the *Gestalt* name. As an explanation of how the perception of individual objects are formed, the Gestalt theory proposes an organization of parts of an image into wholes, based on laws of grouping. Elements in the image are grouped based on proximity, similarity, closure, symmetry, and continuation. These groups themselves can be used as tokens and grouped with the same laws. Unfortunately, the implementation of such observations is difficult, as these laws conflict, even for simple stimuli. In computer vision these ideas can be found in [52] and [97].

The issues we have to tackle in order to generate groupings relate to the choice of such groupings, their discriminative power, their robustness to viewpoint and noise, and their efficient computation.

Here, we address these issues and develop a feature hierarchy which can be used for object recognition of three-dimensional objects from a two-dimensional scene. In this hierarchy, we propose specific groupings based on proximity, parallelism, symmetry, and closure. The detection of these features is performed in an efficient way using *proximity indexing*. First, we generate features with multiple representations to overcome the unreliability of local algorithms during preprocessing, and to handle noise and capture different levels of detail. Later, we merge perceptual similar features at higher levels of the feature hierarchy. As models, we use a set of non-registered views of a 3-D object. While most other systems use spatial correspondences to verify matching hypotheses, we use high level features and their *topological* relationships for the recognition process. These topological relationships can be represented by a graph. The graph vertices represent the features. The edges represent topological attributes between the features, such as inclusion and adjacency. We use the paths in this graph as basic token for the process of structural indexing. Groups of consistent hypotheses represent detected model instances in a scene.

Our proposed groups are not guaranteed to produce “natural” primitives, in the sense of the ribbon of a generalized cone for instance, so we need to validate our choice. This is achieved by demonstrating a recognition system which, even in its early implementation, can recognize an object, known from a significantly different viewpoint, and uses topological constraints as opposed to geometric constraints.

This chapter is organized as follows: we start with a review of previous work in section 4.2. Section 4.3 describes the feature hierarchy, discussing the detection algorithms which we developed to derive the different features from the intensity edges of an image. We then introduce *Proximity Indexing* in section 4.4. In section 4.5 we talk about the motivation of using high level groupings and give an outline of an object recognition system, discuss the matching, and we show some results.

The algorithm presented here is work in progress and therefore we are not able to discuss a complete system. Some topics, such as large data bases, are beyond the scope of this thesis and will be hopefully addressed in future research.

4.2 Related Work

The recognition of “3D from 2D” is the most difficult task, but also the most common. Humans beings are very good at *seeing* three dimensional objects in two dimensional images. Two dimensional projections of the three dimensional world are easily obtainable by cameras, and therefore it is of special interest to be able to analyze them automatically in the way, that three dimensional objects can be recognized.

As most of the previous work is concerned with pose estimation rather than object recognition, we simply refer the reader to [33] for an excellent overview. Other approaches which focus on groups of low level primitives combined with an indexing scheme for the purpose of two-dimensional object recognition can be found in [11, 25, 49, 42]. Using explicitly groupings for recognition was done by the following authors:

Roberts: The study of object recognition of three dimensional objects was launched by the classic work of L. G. Roberts [70]. In his work the scenes are restricted to consist of polyhedral solids with homogeneous surfaces against uniform backgrounds. After edge extraction, linear segments are grouped in order to form polygons. These polygons are matched against model faces based on attributes (like number of corners) to get the right correspondences. The model producing the minimum matching error is chosen. Once an object has been recognized, the lines corresponding to it are removed and the remainder of the scene is analyzed for other models.

ACRONYM: Brooks [12, 13] developed an image understanding system called ACRONYM which uses a restricted class of generalized cylinders (GC) for descriptions of model and scene objects. The model objects are represented by hierarchical graphs of primitive volumes described by GCs. The user constructs a tree for each object, where nodes contain parts of objects represented by GCs, and links represent their subpart relation. The tree is hierarchical in that higher nodes in the tree correspond to more significant parts in the description. The user is also required to construct a model class hierarchy called a restriction graph. This graph contains sets of constraints for different classes of objects and is used later to guide the match between model and scene objects. The root node represents the empty set of constraints. A node is added as a child of another node by constructing its constraint list from the union of its parent's constraints and the additional constraints needed to define the new node's more specialized model class. An arc in the graph always points from a less restrictive model class to a more restrictive one. During the matching process, other nodes are also added to the restriction graph in order to specialize further a given model for case analysis, or to specify an instance of a match of the model to a set of image features.

Ribbons and ellipses are used to describe scene objects. A ribbon is used to describe the projection of a GC body and an ellipse is used to describe the ends (terminators) of the GC. At first, contours of objects are extracted and linked, then ribbons and ellipses are fit to the sets of contours. The descriptions are finally represented by an observation graph whose nodes contain ribbon and ellipse descriptions and links specify spatial relations between nodes.

ACRONYM predicts appearances of models in terms of ribbons and ellipses that can be observed in an image based on viewpoint-insensitive symbolic constraints. A rule-base module followed by a case analysis is used to generate and restrict predictions. Matching is performed at two levels: First, predicted ribbons must match image ribbons, and second, these local matches must be globally consistent.

SCERPO: Lowe's SCERPO [52, 53] system takes a bottom-up approach to object-centered recognition. In SCERPO, objects are represented as polyhedra, or constructions of 3D faces. Image contours are first grouped according to perceptual organization rules, including parallelism, symmetry, and collinearity. From these groupings, simple 3D inferences are made about the 3D contours comprising the object; for example, parallel lines in the image imply parallel edges in the polyhedral object. The 3D inferences are matched against manually identified instances of the properties in the model. Back-projected features are used to verify the object and constrain its position and orientation. Although SCERPO could be applied to

unexpected-object recognition, the complexity of polyhedral models and the simplicity of the indexing features results in large indexing ambiguity. In addition, SCERPO's polyhedral models restrict its recognition domain.

Mundy and Thompson: Mundy and Thompson [58–60] approach the recognition problem with an orientation toward parallel computation by clustering transformations derived from multiple feature assignments. For the determination of a valid model-to-image assignment they use a voting scheme in the transform space. Recognition then becomes the detection of a point in transform space whose coordinates map a significant number of model features to image features. The considerable computation and space requirements such an approach would usually entail are avoided by using a sparse, minimal feature, called the *vertex pair* that is easily extracted to reduce the number of features that can exist in a given scene.

ORA: Ullman and Huttenlocher [39] designed the ORA (for Object Recognition by Alignment) system for 3D from 2D recognition. The ORA system uses one or two pairs of model and image features to compute possible alignments of solid objects with a two dimensional image. The system first extracts features from the intensity edges in an image. These features are then matched against model features, and used to solve for possible alignments of a model with the image. Each alignment is scored by projecting the aligned model features into the image, and counting the number of projected features for which there is a nearby image feature of the same type and similar orientation. Alignments that account for a high percentage of a model's features are kept as correct matches.

ORA uses three types of primitive features: straight edges, corners, and arcs. Groups of connected or nearly-connected primitive features are combined to form alignment features. Each alignment feature defines either a triple of points or an oriented point. Corresponding alignment features in a model and an image are used alone or in pairs to recover the three-dimensional transformation from a model to an image.

Lamdan et. al: A similar technique as described above in section 2.2 by Lamdan *et al.* was extended for the model based recognition of 3D from 2D. In the projective case five points define a basis for the 3D space and four points define a basis for the projective plane. The precomputation, matching, and voting processes work in the same way as described in section 2.2.

SUCCESSOR: In SUCCESSOR, Ponce and Kriegman [47] address the recognition of curved objects. They need a complete model of the solid and use the

edges created by the 3D contours of the objects. The problem is difficult because these contours may be viewer dependent (limbs and cusps) for smooth objects. The equations are solved using the powerful but cumbersome formalism of elimination theory. The results shown are quite promising, but the correspondence issue is still not addressed.

Mohan and Nevatia: Mohan and Nevatia [57] perform segmentation of images based on perceptual clues. They use the resulting segmentation for the detection of buildings in aerial images and for solving the stereo correspondence problem. Their groupings are strongly *natural*, in the sense that the segmentation corresponds to a physical interpretation. A special focus on *natural* descriptions of objects was also done by Rao and Nevatia [69].

Seibert and Waxman: An interesting approach is presented by Seibert and Waxman in [75]. Their algorithm falls in the aspect-graph category. Three-dimensional objects are automatically generated from exploratory view sequences of unoccluded objects. In building the models, processed frames of a video sequence are clustered into view categories called aspects, which represent characteristic views of an object invariant to its apparent position and size. The aspects are used to build the three-dimensional object representation on line. Recognition emerges as the hypothesis that has accumulated the maximum evidence at every moment. The algorithm in its current state assumes no occlusion, and that the object is easy to segment from the background.

Dickinson et al. : Dickinson, Pentland, and Rosenfeld [24] present an approach to the recovery of three-dimensional volumetric primitives (called geons) from a single two-dimensional image. The approach first takes a set of three-dimensional volumetric modeling primitives and generates a hierarchical aspect representation based on the projected surfaces of the primitives. From a region segmentation of the input image, they present a novel formulation based on the grouping of the regions into aspects. Once the aspects are recovered, they use the aspect hierarchy to infer a set of volumetric primitives and their connectivity. They show some results on some simple scenes. The applicability to a large data base and noisy images still has to be demonstrated.

Forsyth et al. : The authors in [31] are discussing the use of invariant shape descriptors that are unaffected by object pose, by perspective projection, and by the intrinsic parameters of the camera. These descriptors can be constructed using the

methods of invariant theory. The results are demonstrated on a model data base of fifteen conic objects.

other systems: For the case of two-dimensional images and three-dimensional models consisting of point features Clemens and Jacobs demonstrate in [20] bounds on the space required for indexing and on the speedup that such indexing can achieve. They find that indexing can produce only a minimal speedup on its own. However, when accompanied by a grouping operation, indexing can provide significant speedups.

Ullman and Basri propose in [94] an approach to recognition in which a three-dimensional object is represented by the linear combination of two-dimensional images of the object. They discuss the handling of sharp as well as smooth boundaries. In their paper they assume that the correspondences are given.

There exist other systems performing model directed object recognition which put the emphasis on finding the accurate position of models in the scene. See for example [23, 37, 47, 91]. They assume the correspondence problem is solved. Generally known as the *perspective n-point problem* the idea is the following: Given a set of points with their coordinates in an object centered frame and their corresponding projections onto an image plane and given the intrinsic camera parameters, find the rigid transform between the object frame and the camera frame. Although this can be part of an object recognition system, the solution might well be used for determination of exterior camera parameter (position and orientation). Since our goal is to recognize an object in a scene, we are concerned about *which* model we find in a scene and *where* approximately it is located. The accurate position can be computed by refining these results.

Other systems focus on the perceptual grouping. As an example we mention two interesting approaches:

- Huttenlocher and Wayner [40] describe an algorithm to find the largest convex groupings of line segments. The advantage of such an approach is that the number of groups generated is smaller than the number of original primitives. The role of such groupings in recognition has yet to be demonstrated, however.
- A different approach of perceptual grouping is taken by Sha'ashua and Ullman [76]. They focus on the saliency of structure in an image. They present a saliency measure based on curvature and curvature variation, and they show examples of cluttered scenes where their results correspond to the human focusing mechanism.

Table 4.1: 3D from 2D

(b) The same Detail in another View

Figure 4.1: The Problem of Matching Local Features

Here now we face a more difficult problem. The above cited co-curvilinearity is no longer a reliable criterion for a stable grouping strategy. An example can be seen in figure 4.1. Figure 4.1(a) shows a part of a house and a closeup view. Figure 4.1(b) shows the same part only from a different view and under slightly different illumination. It is obvious that it is extremely difficult, if not impossible, to generate features which are only based on connectivity which enable us to find the *same* representation for these two regions in order to find a match. The solution which we pursue in this chapter is to go to higher level groupings which take into account other grouping criteria besides co-curvilinearity, such as parallelism, closure, and symmetry.

We now explain the steps involved in going from an image to a high level representation of it in terms of “perceptual” groups. This chain of processing is sketched in Figure 4.2. We start with an image. In the *preprocessing* stage we reduce the amount of data: Starting from images we compute curves, which consist of linked edgels.

Figure 4.2: Hierarchy

(d) One U-Shape

Figure 4.3: Multiple Interpretations

To process all the results from the previous stage, we use a coarse approximation to represent a *high level grouping*, namely its convex hull. With the convex approximations we build a topological graph. The graph edges are defined by topological relationships such as adjacency or inclusion.

The following sections focus on the details of the perceptual hierarchy. We start with the elementary primitives, the edgel curves, and discuss the implementation of the different grouping rules:

- the co-curvilinearity criterion produces *super segments*,
- the parallelism criterion produces *parallels*, which consist of two parallel line segments,
- the symmetry criterion produces either *parallel symmetries* or *skewed symmetries* which are computed from parallels or super segments respectively,
- and the closure criterion produces *U-Shapes*, *closed curves*, and *skewed symmetries*.

Finally we discuss the computational efficiency of our perceptual grouping algorithms. In computer vision a lot of research was focused on computing perceptual groups (see *e.g.* [36, 43, 52, 57, 47, 71, 73, 95]). Most of these algorithms focus on the detection of *all* perceptual groups by either assuming perfect data, or by applying exhaustive search. Our algorithms try to make a compromise: we do not assume perfect data and therefore we find most (but not necessarily all) perceptual groups. But on the other hand we do this in an efficient way by using *proximity indexing*.

4.3.1 Preprocessing

We first apply an edge detection algorithm on the image (we use the Canny edge detector [16]). The resulting edgels are further linked into *curves*. Curves are then approximated with a line fitting algorithm to compute the polygonal approximations. Instead of just using one representation we approximate each curve with a set of line fitting tolerances to get a robust representation. For every approximation, we then collapse vertices, so that vertices which are close together (typically three to five pixels) are collapsed into one vertex.

4.3.2 Super Segments

Idea Since we want to handle occlusion, we do not expect to obtain complete boundaries in our images, but only portions of them. On the other hand, individual segments are too local to be useful as matching primitives. Grouping a fixed number of adjacent segments provides us with one of our basic features, the super segments.

Implementation The computation of super segments is the same as described in [78]. Connected linear segments form chains of adjacent segments. The segment chains provide the super segments by grouping a fixed number of adjacent segments.

(Because of the branching of the polygonal approximations we generate super segments exhaustively by generating them from all possible segment combinations.) We typically generate super segments of cardinality three to six.

4.3.3 Parallels

Idea A *parallel* consists of two linear segments s_1 and s_2 . Both line segments have approximately the same orientation. We require that the two segments overlap, which means that the normal projection of s_1 on s_2 , or the normal projection of s_2 on s_1 not be empty. Furthermore we do not want the aspect ratio of the parallel to reflect elongation, with $\text{ar}(p) = (\text{length}(s_1) + \text{length}(s_2))/\text{distance}(s_1, s_2)$. Typically we require $\text{ar}(p) > 0.5$.

Implementation The acquisition of the parallels is performed in two steps by using proximity indexing (for details, see Section 4.4).

1. All segments are recorded using the quantized orientation as the key. We use δ as quantization (typically $\delta = 20^\circ$). Using proximity indexing we are guaranteed to find parallels which are at most $\delta/2$ apart and we get some parallels with an enclosed angle between $\delta/2$ and δ .
2. For every linear segment, the possible candidate parallels are retrieved and verified with respect to aspect ratio and overlap. Segment pairs which meet these constraints generate parallels.

4.3.4 Symmetries

A symmetry is defined as a one-to-one mapping between the points of two curves, with the symmetry axis defined as the locus of the mid-point of the straight lines joining a point on one curve to its image in the other [57]. Ulupinar and Nevatia have proposed two specific symmetries, namely skewed and parallel symmetries [95]. Whereas they use them to infer surface orientation, we consider them as a general feature of an object.

Parallel Symmetries

Idea Given two curves $X_i(s) = (x_i(s), y_i(s))$, for $i = 1, 2$, parametrized by arc length s , and $\theta_i(s) = \arctan((dy_i(s)/ds)/(dx_i(s)/ds))$. $X_1(s)$ and $X_2(s)$ are said to be parallel symmetric [95] if there exists a point-wise correspondence $f(s)$ between them such that $\theta_1(s) = \theta_2(f(s))$ for all values of s for which X_1 and X_2 are defined,

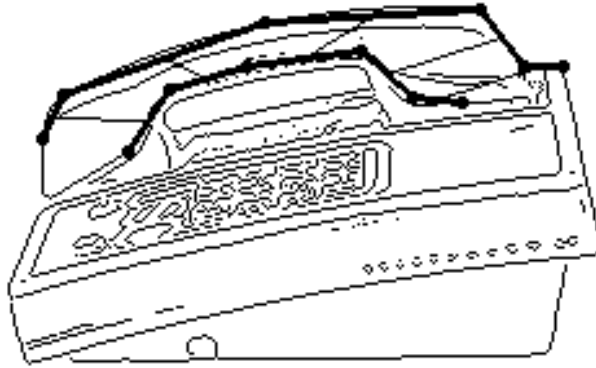


Figure 4.4: Example of a Parallel Symmetry

and $f(s)$ is a continuous monotonic function. We only consider the special case where $f(s)$ is a linear function.

Implementation Parallel symmetries are retrieved by finding proximate parallels. We do not use the super segment approach, because we would depend on the cardinality of the super segments. By using the parallels as the building blocks, we can use proximity indexing to find parallels which share the same vertices. The acquisition of the parallel symmetries is performed in two steps.

1. We record every parallel twice. One time with the sorted list of the vertex coordinates of one side as key, the other time with the sorted list of the vertex coordinates of the other side as key.
2. For every parallel, the possible neighbor parallels are retrieved. The groups of adjacent parallels generate parallel symmetries.

One example is shown in Figure 4.4.

Skewed Symmetries

Idea In a skewed symmetry, the point-wise correspondence is such that the axis of the symmetry is straight, and the lines of symmetry from a constant angle (not necessarily orthogonal) with the axis of symmetry. Skew symmetry was first proposed by Kanade [43] and used in the analysis of scenes of polyhedral objects. An example is given in Figure 4.5(a).

The detection of skew symmetry for curves was done by Ponce [47] and Saint-Marc and Medioni [73], but these methods are quite sensitive to noise.

(b)

Figure 4.5: Examples to (a) skew symmetry with curved contours, (b) and skew symmetry with straight contours. The bold curves are axis of symmetry and the dotted lines are lines of symmetry.

In our system, we are interested in symmetries between line segments. Considering all possible symmetries between line segments as proposed in [36] is expensive.

Our approach is based on finding skew symmetries between super segments. An example of skew symmetry for straight line segments is given in Figure 4.5(b). As we show in the Appendix, we have $\Delta = |\alpha^i - \beta^i| \leq 2\theta$, where θ is the skew angle. This allows us to define a local signature for a super segment, namely the angles, with which we can find possible symmetry candidates by indexing. This avoids the expensive comparison of all pairwise super segments. To test whether two super segments are skew symmetric, we have to test the following:

1. The difference between the corresponding angles must be smaller than $2\theta_{\max}$.
2. The symmetry axis has to be straight.

Using proximity indexing allows us to obtain an efficient algorithm.

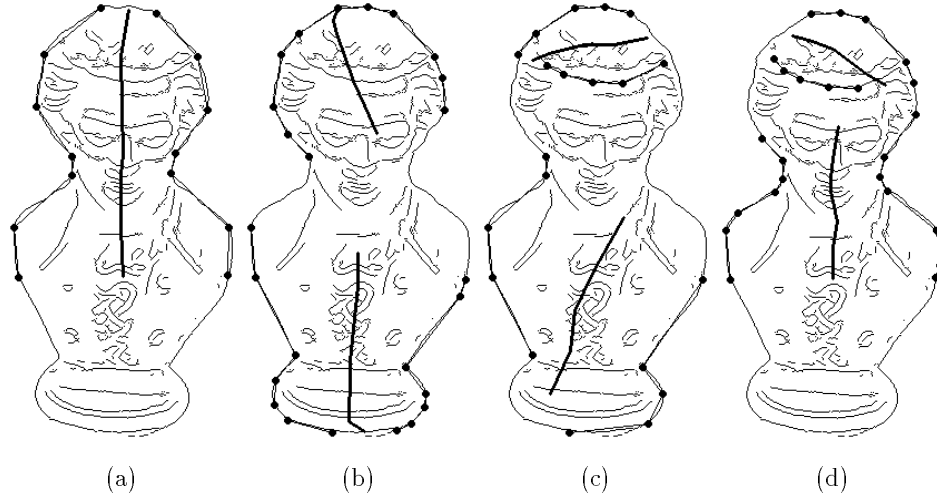


Figure 4.6: Skewed Symmetries and the Corresponding Axes in Mozart Bust

Implementation Skewed symmetries can be retrieved by finding pairs of super segments with angles of opposite sign. The detection of the skewed symmetries is performed in three steps:

1. We record every super segment using the list of the curvature angles as a key. To allow the guaranteed detection of symmetries with up to skew angle θ_{\max} , we choose as the quantization interval for the super segment angles $2\theta_{\max}$.
2. For every super segment, we use the list of opposite sign curvature angles to retrieve possible symmetry candidates.
3. Super segment pairs which generate symmetry hypotheses have to be checked to see whether the corresponding symmetry axis is straight. We test this by requiring the scatter matrix of the middle points of all corresponding vertices to have a high eccentricity (they lie all along a line).

We show an example for a set of detected symmetries in Figure 4.6.

4.3.5 Closures

Lowe [52] makes the following statement: *There is a tendency for curves to be completed so that they form enclosed regions.* Based on this statement, Mohan and Nevatia [57] developed the idea to close symmetries at their ends to obtain so called *ribbons*, which form enclosed regions. They use these ribbons to segment images.

We want to use closures as features, which do not necessarily have any physical interpretation in the image. At the moment we compute closures from U-Shapes, from closed curves and from skewed symmetries.

Closure from U-Shape

Idea A parallel which is closed at one side by a linear segment is a strong indication that a rectangular structure is at hand where one side could not be detected. We therefore assume that we found a closed contour.

Implementation U-Shapes can be found by indexing over the vertex pairs of parallels and trying to find a segment which forms a U-Shape with the parallel.

1. We record every parallel twice using once the quantized vertices of one “side” of the parallel and then other vertices as keys, where we record the parallel.
2. For every linear segment we use the list of the endpoints to retrieve possible U-Shape candidates.
3. If the angle between the parallel and the segment is $90^\circ \pm 30^\circ$ we generate a new U-Shape.

Closure from Curve

The obvious form of a closure occurs if we have a closed curve. To detect a closure based on a curve we allow the gap between start and end of the curve to be 5% of the arc length of the curve.

Closure from Skewed Symmetry

Idea We adopt the idea that a segmentation into parts should be done at negative minima of curvature from Rom and Medioni [71]. Such “a part” is used in our system as a closure.

Implementation For every skewed symmetry we traverse the angles of one of its super segments (the other super segment just has the skewed mirror angles). Whenever we encounter a sign change of consecutive angles, we “break” the symmetry at this point and define the symmetry up to this vertex (together with the corresponding vertex of the other super segment) as one part. Applying this step iteratively, we generate alternating convex and concave parts. We use the convex parts to create closures. An example can be seen in Figure 4.7.

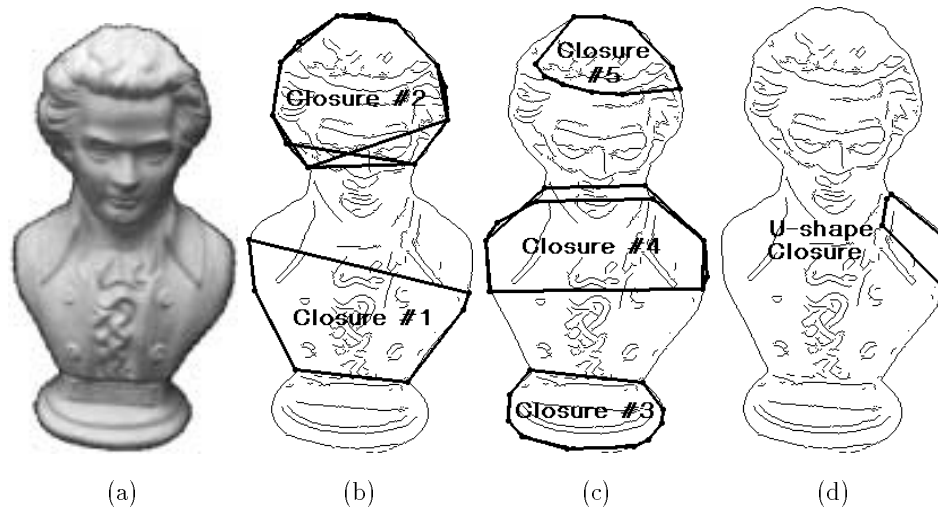


Figure 4.7: Detected Closures in Mozart Bust: (a) shows the original image, (b) and (c) show five different closures from skewed symmetries, and (d) shows a closure from U-shapes, which is a strong hint of a rectangular structure. We only display the convex hulls.

4.4 Proximity Indexing

We encounter two questions in dealing with the generation of perceptual groups:

1. How do we cluster initial features based on their characteristics and geometric relationship in an efficient way, in order to get a combination which represent a *higher level* features?
2. How do we merge perceptual features to avoid multiple features with the same perceptual content?

Both questions point towards a difference measure. Given a feature, we define its characteristics as a *vector* of attribute values (e.g. eccentricity, orientation, location). The difference between two features is then defined as the pairwise difference between the attribute values. If all the differences are small, then the two features are similar.

Given a set of features, how do we find the ones with similar attribute values? Traditional search methods which compare every possible pair are very time consuming. Therefore recent vision systems [49, 81] have used indexing to find corresponding features with similar characteristics. The big issue in indexing is the crucial parameter for the quantization size and the question: “What happens to values which are close, but fall in different quantization intervals?” Two features match only in the

case when both keys are exactly the same. This means that all the pairwise values have to fall into the same quantization intervals. Suppose the range is quantized into intervals of the the same size q . The key length is n . Each value, v , is then assigned a key based on which interval v falls into, and v is corrupted by a random additive term bounded by ε ; then the probability that a corrupted vector is assigned the same entry k as the original one is

$$p^n(k|v) = (1 - \varepsilon/q)^n.$$

Looking at this equation, it is obvious that the probability of matching long keys decreases rapidly (see also section 2.5).

As Flynn and Jain [30] point out, it is essential to have an indexing scheme that preserves proximity in the key values. So far, two strategies based on indexing have been used to deal with this problem: large bucket size and searching of neighboring bins. While large bucket size is based on the hope that “less values will fall into the incorrect bin”, the search of neighboring bins has an exponential complexity with respect to the number of false value matches (when we have to check for every bin both neighbor bins, the complexity is $O(2^n)$ for a key length of n).

We propose an alternative approach: Instead of using all features with all quantizations as a large alphabet, we break the features apart and use indexing on every value separately.

Encoding: To assure proximity we store the feature for every value v twice:

1. under the key $[\lfloor v \rfloor_q, \lceil v \rceil_q]$
2. under the key

$$\begin{cases} \left[\lfloor v \rfloor_q^+, \lceil v \rceil_q^+ \right] & \text{if } v - \lfloor v \rfloor_q > \lceil v \rceil_q - v \\ \left[\lfloor v \rfloor_q^-, \lceil v \rceil_q^- \right] & \text{otherwise,} \end{cases}$$

with

$$\begin{aligned} \lfloor v \rfloor_q &= q \lfloor \frac{v}{q} \rfloor & \lceil v \rceil_q &= \lfloor v \rfloor_q + q \\ \lfloor v \rfloor_q^+ &= \lfloor v \rfloor_q + \frac{q}{2} & \lceil v \rceil_q^- &= \lfloor v \rfloor_q - \frac{q}{2} \\ \lfloor v \rfloor_q^- &= \lfloor v \rfloor_q + \frac{q}{2} & \lceil v \rceil_q^+ &= \lfloor v \rfloor_q - \frac{q}{2}. \end{aligned}$$

Retrieval: The retrieval of a feature f is broken up into the retrieval of every value. Every value is quantized twice (see above) and the stored features for both intervals are retrieved and combined into one set. For all values we get such a feature set. The intersection of all these sets results in the features which are close to f in *all* their values. Due to the interlaced quantization we can guarantee to retrieve all features with $|v_{\text{stored}} - v_f| < \frac{q}{2}$, and we get some feature matches with

$\frac{q}{2} < |v_{\text{stored}} - v_f| < q$. The intersection process can be sped up by intersecting the sets in increasing cardinality.

Complexity: What do we pay for using proximity indexing compared to traditional indexing? Let n be the number of features to store. We assume every feature has as a key a vector k with values v_i with $i = \{1, 2, \dots, p\}$, p is the number of attribute values per feature. Every value is quantized in Q intervals. Furthermore we assume that the features are equally distributed over the table which consists of r records. Every record has a key and a list of h entries. Therefore, for traditional indexing there are $h = \frac{n}{r}$ entries per record, and for proximity indexing $h = \frac{2pn}{r}$ entries per record. In traditional indexing $r_{\text{max}} = p^Q$ and in proximity indexing $r_{\text{max}} = pQ$.

Conventional indexing has a space requirement of $O(n)$, the same is true for proximity indexing.

Looking at the time complexity, we have to distinguish between the encoding step (building the index table) and the retrieval step (retrieving corresponding features from the table).

1. The time complexity for encoding with traditional indexing is $O(n)$, whereas the complexity for encoding in proximity indexing depends on sorted or unsorted encoding. As it will become clear in the discussion of the retrieval step, sorted encoding results in a faster retrieval. By sorted encoding we talk about sorted entries in each record. The sorting can be performed by using an arbitrary but stable sorting function (e.g. the creation date of the feature). Using a binary tree for the storage of the entries, the complexity of the encoding of n features is $O(n \log(\frac{n}{r}))$.
2. The retrieval step requires $O(n)$ for traditional indexing. The retrieval of features for proximity indexing requires the intersection of sets of features. Intersection of two sets of cardinality c has a complexity of $O(c)$ when the sets are sorted. The cost of intersection of two unsorted sets is $O(c \log(c))$. Therefore, when the features are encoded in a sorted way, the retrieval step has a cost of $O(\frac{n^2}{r})$, otherwise, in the unsorted case, the complexity is $O(\frac{n^2}{r} \log(\frac{n}{r}))$.

A comparison of the different approaches is summarized in the following table:

	search	indexing	proximity indexing
preserves proximity	yes	no	yes
encoding complexity	0	$O(n)$	$\begin{cases} O(n) & \text{if entries are unsorted} \\ O(n \log(\frac{n}{r})) & \text{if entries are sorted} \end{cases}$
retrieval complexity	$O(n^2)$	$O(n)$	$\begin{cases} O(\frac{n^2}{r} \log(\frac{n}{r})) & \text{if entries are unsorted} \\ O(\frac{n^2}{r}) & \text{if entries are sorted} \end{cases}$

All tables mentioned in this section are implemented as hash tables.

4.5 Representation and Matching

Given a set of features and the topological relationships between them, a natural representation of this structure is a graph. First we give some definitions:

- A **graph** G is defined by the pair (V, E) ; $V = \{v_0, v_1, \dots, v_n\}$ is the set of vertices, and $E = \{e_0, e_1, \dots, e_m\}$ with $e_i = (v_{i1}, v_{i2})$ is the set of edges.
- A **subgraph** of $G = (V, E)$ is defined as a graph $G' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$.
- Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are **isomorphic** if there exists a bijection $f : V_1 \rightarrow V_2$ such that $(u, v) \in E_1$ if and only if $(f(u), f(v)) \in E_2$.
- A **path** P of length k in G is a sequence $\{v_0, v_1 \dots v_k\}$ of vertices such that $(v_{i-1}, v_i) \in E$ for $i = 1, 2, \dots, k$.
- When the edges of a graph are labeled we talk about a **colored** or **labeled** graph.
- The **adjacency-matrix** representation of a graph G consists of a $|V| \times |V|$ matrix $A = (a_{ij})$ such that $a_{ij} = \text{label}(v_i, v_j)$.
- The adjacency-matrix of **cardinality** n is A^n with $A^0 = I$, $A^n = A^{n-1} * A$. It represents the set of paths of length n between any two vertices.

Two edge labels are multiplied by an order preserving multiplication. That means: $l_1 * l_2 = l_1 l_2 \neq l_2 l_1$. The entry $a_{ij} = l_1 l_2 = \text{label}(v_i, v_k) * \text{label}(v_k, v_j)$ represents a path from v_i to v_j through v_k with the labels l_1 and l_2 respectively.

The addition between two labels is not order preserving. The entry $a_{ij} = l_1 l_2 + l_3 l_4$ represents the fact that there are two paths from v_i to v_j with either the labels l_1 and l_2 or l_3 and l_4 .

(d) Adjacency Matrix with
Cardinality 2

Figure 4.8: Example 1

As an example see figure 4.8: We have four CAs: C_1 , C_2 , C_3 , and C_4 . C_1 is adjacent to C_4 , C_3 is inside of C_1 , C_2 , and C_4 . In addition C_3 is adjacent to C_2 . C_2 is adjacent to and inside of C_1 . The corresponding graph can be schematically represented as in figure 4.8(b). Another example is shown in figure 4.9(a) with the corresponding graph in figure 4.9(b).

By representing the scene and the models with graphs, the matching becomes a task of subgraph isomorphism. Which subgraphs of the scene are isomorphic to which subgraphs in the models?

(d) Adjacency Matrix with
Cardinality 2

Figure 4.9: Example 2

In computer vision, the matching of graphs is widely used (see *e.g.* [64, 26, 56, 67]). The worst case complexity of the subgraph isomorphism problem is NP-complete. Therefore several heuristics were developed to improve the average complexity for specific cases (see *e.g.* [5, 22, 93]). Despite all the previous research, we could not find any algorithm which would perform with reasonable complexity for graphs consisting of a hundred vertices or more. Furthermore, no research is known to the author of any efforts to find subgraphs in colored graphs. Our goal is to find large subgraph isomorphisms, which are likely to represent detected models in a scene. We believe that we can use structural indexing to find corresponding subgraphs.

What are the structural tokens on which we index? Using the CAs only would be very expensive. In our implementation they are not labeled and therefore they have little information content. All the “color” of the graph lies in the relationships between the CAs, namely the edges in the graph.

The idea is to find all paths of length k (corresponding to $k + 1$ connected CAs) and to cluster these to find the largest consistent group of such paths. How many

such paths exist in a graph? The worst case is a complete graph where every pair of vertices is connected by an edge. In a complete connected graph consisting of $|V| = m$ vertices, there are $m - 1$ outgoing edges at every vertex ($|E| = (|V| - 1)!$). Therefore the number of paths of length k is $m(m - 1)(m - 2)\dots(m - k)$. This corresponds to an upper bound of $O(|V|^{k+1})$ in the number of paths. A more realistic assumption is that there are only a constant number of outgoing edges at every vertex. This results in an upper bound of $O(|V|^k)$ in the number of paths.

On one hand we are interested in using long paths to be as discriminative as possible, on the other hand the number of possible paths in a graph grows exponentially with k . Another consideration for k is the size of the corresponding subgraphs. Choosing a large k can result in not detecting a subgraph which has less vertices than k .

In our implementation we use the path length $k = 2$. This allows us to exploit the discriminative power of three connected CAs. At the same time the number of paths has the worst case complexity of $O(|V|^2)$ (assuming a constant number of outgoing edges at every vertex). The clustering of corresponding paths enables us to find the corresponding subgraphs with more than 3 vertices.

The computation of the paths is straightforward. As shown in figure 4.8 the graph in (b) can be represented by its adjacency matrix in (c). The adjacency matrices of higher cardinality can be computed easily (see figure 4.8(d)).

The representation of an objects works as illustrated in figure 4.10. Every view of a model is processed in the following way:

1. The feature hierarchy is computed.
2. The CAs are used to create the topological graph.
3. All paths (in our implementation of length 2) are computed.
4. Every path is encoded and stored in a data base. To encode a path we take the code of all pairs of CAs. We use the following attributes to encode a pair of CAs:
 - the label of the connecting edge,
 - when the two CAs are adjacent, the percentage of common boundary
 - when the two CAs are intersecting, the percentage of area intersection.

All numerical values are quantized in a coarse way to allow significant deviations due to viewpoint change and noise. The typical quantization in our implementation for all numerical values is 20%.

Figure 4.10: Representation of a Model

The generation of the hypotheses proceeds in a similar way (see figure 4.11):

1. the feature hierarchy of the scene is computed,
2. the CAs are used to create the topological graph,
3. all the paths are computed,
4. every path is encoded,
5. and retrieves from the data base the stored model paths which have the equivalent code.

The retrieved hypotheses are equivalent to subgraph isomorphisms of path length 2 between the different model-views and the scene. In the verification step we cluster the hypotheses in order to get larger corresponding subgraphs which are likely to represent an instance of a model in a scene. Two hypotheses are consistent when the following rules apply:

1. They share at least one corresponding CA pair.

Figure 4.11: Retrieval of Hypotheses from a Scene

2. No contradiction occurs. That means that the combined number of vertices and edges of the two paths in the model-view have to be the same as in the scene.
3. The connectivity has to be preserved. When two vertices are connected in one subgraph they have to be connected with the same label in the corresponding subgraph.

In this case, the combined hypotheses form a new hypothesis. These clusters grow iteratively until no further consistent hypotheses pairs can be found.

An example can be seen in the matching between graph 1 and graph 2 of figures 4.8(b) and 4.9(b). We are matching paths of length 2. For simplicity, a path is only encoded by all its edge labels (maximal 3). For example, the path C_3 to C_1 via C_2 has the code **bbi** (or $\mathbf{b}^2\mathbf{i}$), because the edge from C_3 to C_2 has the label **b**, the edge from C_2 to C_1 is labelled with **b** and C_3 to C_1 is labelled with **i**.

It is obvious that loops (closed circuits) of length 2 do not contribute to the structural information in the matching process. This fact is already implicit within an undirected edge. After storing graph 1 without the loops (which are located in the diagonal of the adjacency-matrix), the table consists of three buckets as illustrated in figure 4.12(a).

(b) Generated Hypotheses

Figure 4.12: Corresponding Subgraphs

The hypotheses retrieval step of graph 2 leads to the two hypotheses listed in figure 4.12(b). The verification step tries to cluster these hypotheses. This is done by pairwise checking all hypotheses if the above mentioned consistency rule applies.

The consistency check of h_1 and h_2 fails because rule 3 is violated. The connection between C_3 and C_4 has the label i and the corresponding connection between D_5 and D_2 has the label a .

Therefore we find two subgraph isomorphisms as displays in figure 4.13. The corresponding vertices are illustrated by different shades of gray.

4.6 Results

4.6.1 Buildings

In figure 4.14 we show an example of the performance of our system. Figure 4.14(a) shows the scene. As a model we use one view of a building which is displayed in (b). In (c) and (d) the edgel images are respectively displayed. The perceptual grouping results in approximately 100 CAs for the model and 400 CAs for the scene. It is not necessary to use all of these CAs for the recognition because most of the smaller CAs correspond to small object areas (such as subparts of windows) which are not relevant for the recognition process. We therefore used only the 40 largest CAs of the model and the 160 largest CAs from the scene.

The system finds a large set (approximately 3000) of hypotheses corresponding to matching groups of three CAs. Some examples are shown in figure 4.15 and 4.16. After clustering the consistent hypotheses, the system finds the largest cluster shown in 4.17. The detected cluster consists of seven corresponding CA-pairs which cover

(b) h_2

Figure 4.13: Corresponding Subgraphs

most of the visible area of the model in the scene. Looking at some of the corresponding CAs illustrates the strength of the system to deal with noise and a viewpoint change.

4.6.2 Discussion

What would have happened if we would have taken shorter or longer paths as basic matching primitives? Given k as the path length. Then c_k is the number of CA pairs in a path consisting of $k+1$ vertices, with $c_k = \frac{(k+1)k}{2}$. In section 4.5 we talked about the attributes which we use to encode a pair of CAs: the label of the connecting edge, the percentage of common boundary, and the percentage of area intersection. The label can have four different values: nil, adjacent, inside, or adjacent and inside. We further mentioned that we quantize the last two values in five quantizations of

20% each. That means we have $a = 4 * 5 * 5 = 100$ different codes to encode a pair of CAs. Therefore the number of available path codes of length k is

$$D_k = a^{c_k} = a^{\frac{(k+1)k}{2}}.$$

D_k is a measure for the discriminative power of an encoding scheme. The larger D_k , the larger the code alphabet, the more discriminative power a feature has. In section 4.6.1 we have $k = 2$ and therefore $D_2 = 100^3 = 1000000$.

The trade-off lies in the generation of the matching primitives versus the generation of the hypotheses with respect to the discriminative power. Taking a short path length results in a low number of paths to generate. For $k = 1$, the complexity of the number of paths is $O(|V|)$. On the other hand the discriminative power of these paths is lower. For $k = 1$, $D_1 = 100^2 = 10000$. Because the number of paths decreased by one order of magnitude and the discriminative power decreased by two orders of magnitude, the number of generated hypotheses h is in general larger than in our example in section 4.6.1. Because the clustering of the hypotheses has a complexity of $O(h^2)$, the matching and verification for $k = 1$ is slower than for $k = 2$.

Taking a long path length results in a large number of paths. For example, for $k = 4$, the complexity of the number of paths is $O(|V|^4)$, and $D_4 = 100^{10} = 10^{20}$. The number of generated hypotheses will be minimal due to such a high discriminative power, and therefore the final clustering will be very fast. But computing $O(|V|^4)$ paths requires a high space complexity which may be prohibitive.

The right way to proceed is to increase the value a . This can be done by improving the encoding scheme. So far we disregard the properties of the CAs. We regard them all as equal. In our final chapter 6 we address the issues how such a future extension could be developed.

4.7 Conclusion

In this chapter, we have developed an approach to use perceptual organization for the purpose of object recognition, and show some promising results. Our perceptual grouping is purely data driven. We do not try to resolve any ambiguities and the groupings do not necessarily lead to a single physical interpretation. By using the feature groups for recognition purposes we make effective use of the underlying organization.

- Our system deals with *topological* relations, not with *spatial correspondences*.
- By using a set of different views to represent a model we can deal with *incomplete model descriptions*.

Our future work aims at answering the following questions:

- What happens when the system does not find corresponding high level groupings (e.g. due to heavy occlusion)? We want to focus on this point by developing a multilevel matching, which allows the system to “fall back” on lower level features in order to find correspondences.
- We want to extend the feature hierarchy by including perceptual organization between high level features. So far we use only proximity and closure to generate high level groupings. We are investigating the possibility of including symmetry and parallelism.
- There are several other features and grouping strategies which we ignore so far: continuation, texture, saliency ... Including these features would enrich the descriptive and discriminative power of our feature hierarchy.

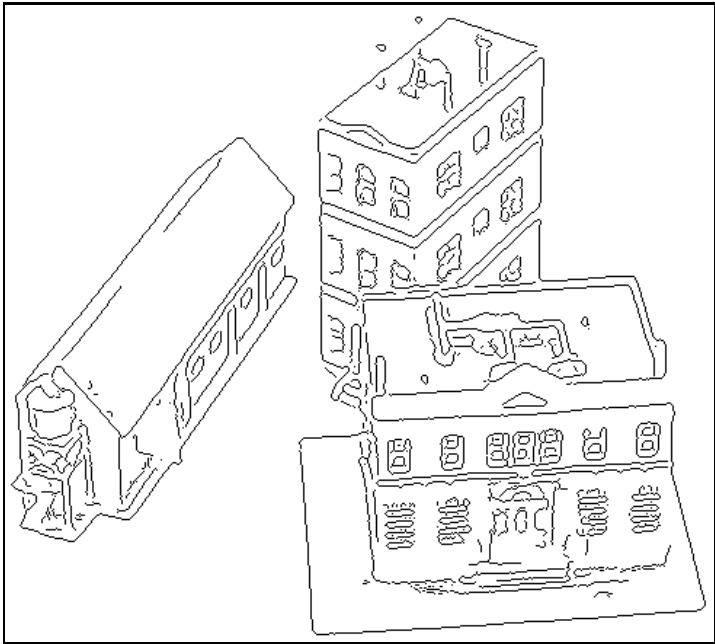
Our work and the corresponding results in this paper have demonstrated the viability of this approach.



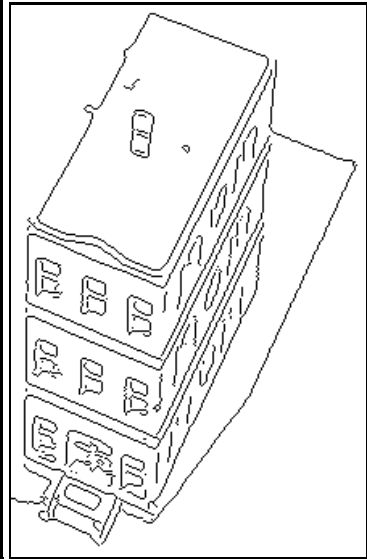
(a) Scene



(b) Model



(c) Edges of Scene



(d) Edges of Model

Figure 4.14: Building Scene

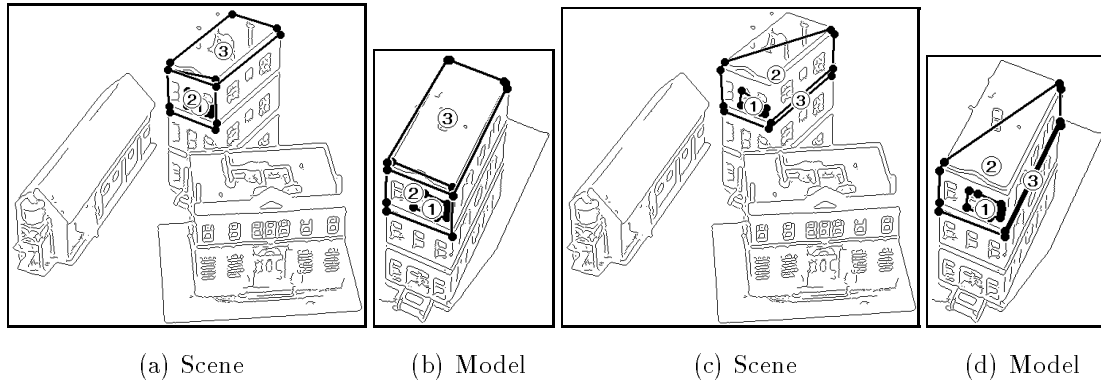


Figure 4.15: Hypotheses Examples

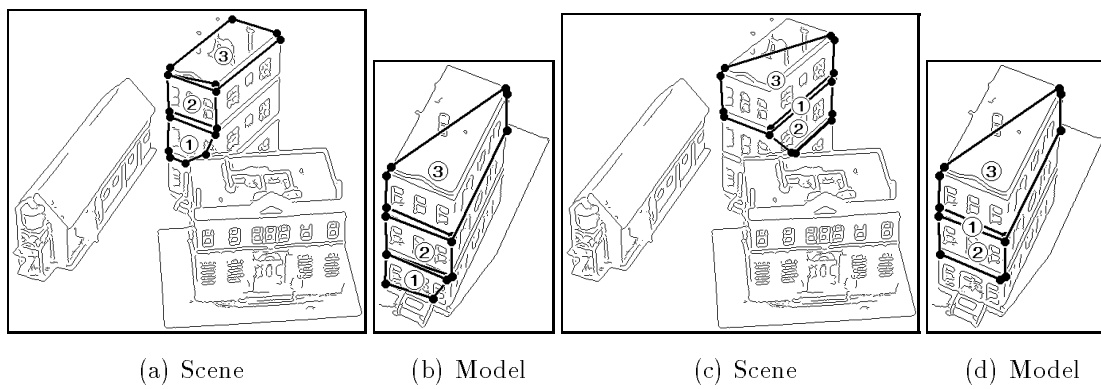
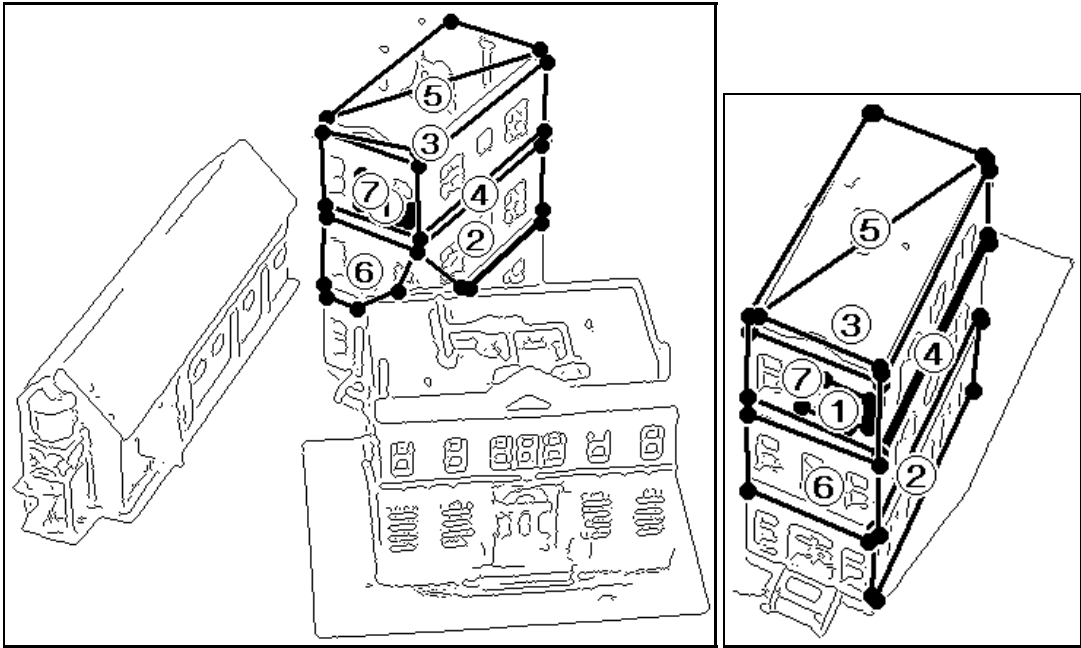


Figure 4.16: Examples of Incorrect Hypotheses



(a) Scene

(b) Model

Figure 4.17: Result

Chapter 5

An Application: The Drop-Off Problem

5.1 Introduction

In this chapter we look at a specific application of structural indexing, namely the localization of an observer in an unfamiliar environment. We demonstrate how an original three-dimensional problem can be reduced to two dimensions and solved with a system similar to the one presented in chapter 2.

Navigation using maps requires the frequent updating of the location of an observer with respect to the map. Humans performing this task use a large number of different problem solving techniques, both expectation driven, where a landmark is selected on the map and searched for, or data driven, where image features are extracted first and matched against the map.

The problem most often addressed by previous researchers is that of updating the current position, given a good initial solution. This can be solved by correcting (small) errors between the predicted and observed aspects of the scene.

Here instead, we study the “drop-off” problem. An excellent overview of the whole problem is given in [89]. The authors describe a preliminary computational model for the drop-off problem (the name *drop-off* comes from the case in which an observer is “dropped off” into a unfamiliar environment and has to orient itself). The observer stays at one position and tries to find its location based on visible landmarks and salient features. In our approach, the observer tries to establish its location based on the curve described by the panoramic horizon (as explained later) which is visible from his viewpoint.

It should be noted that people experience serious difficulties in solving such localization problems, leading many researchers to suggest that traditional object recognition strategies are unlikely to succeed [89]. This is based on the observation that the combinatorics of the problem are very unfavorable, as the shapes are complex and the number of different aspects is extremely large.

In this chapter, we challenge this view and propose that a table-based matching strategy [78, 81] can overcome the limitations mentioned previously. We propose to extract from many locations in the map the panoramic horizon curves¹ which would be observed by the observer at each location. Such curves are encoded and stored in a table. To locate an unknown location, we first extract the panoramic horizon curve of the unknown location. Then we approximate it by a family of polygons with different line fitting tolerances. By using indexing into the table, we retrieve candidate locations. The correct candidate (the closest one) is found by applying further geometrical constraints in the verification step.

It should be clear to the reader that this approach is not guaranteed to provide a unique answer, as it is easy to come up with counter examples (planar or repetitive environment), but we show that it gives excellent results in complex environments. We start by reviewing some of the relevant work in this area, then precisely define the terms we will be using, and clearly state the problem. We then proceed to explain our approach by discussing the basic idea, and present the algorithm. We validate our claims by showing some results from a real map.

5.2 Related Work

Given a map and a certain view from the viewpoint of an observer, the question is: what is the exact position of the observer? Humans are able to perform this matching task by trying to find correspondences between significant landmarks (rivers, mountains, roads) in the surrounding environment and the corresponding features on the map. From a computational view, we want to do the same. But what is the definition of a significant landmark? How do we build a feature detector for the various important clues? Several systems were developed to deal with this problem. Most of these systems assume a known initial position and try to update their location while they move in order to perform correct navigation.

Thorpe *et al.* [90] present results on the visual navigation for mobile robots in outside environments. They discuss two algorithms: color vision for road following, and 3D vision for obstacle detection and avoidance. The resulting system is able to navigate continuously on roads while avoiding obstacles.

Zheng *et al.* [98] introduce an approach to build qualitative descriptions of scenes along a route, which is used in route recognition by a mobile robot. The robot uses autonomously selected landmarks from the panoramic views for navigation. The

¹These features will be fully defined later. Briefly, they represent the crest line perceived by the observer as he completes a full 360° view in place.

landmarks are detected by using an algorithm which detects “unique patterns” in the images.

Arkin *et al.* [2] explore several visual strategies for the navigation of a mobile robot. These include a line-finding algorithm for path following, a multiple frame depth-from-motion algorithm for obstacle avoidance, and the relationship of schema-based scene interpretation to mobile robotics, especially regarding vehicle localization and landmark recognition. They present results of actual robot navigation in an outdoor environment.

Nasr *et al.* [61] present a landmark recognition technique based on a perception-reasoning-action and expectation paradigm of an intelligent agent. It uses extensive map and domain dependent knowledge in a model-based approach. They present examples using real ALV (Autonomous Land Vehicle) images.

Levitt *et al.* [50] developed a theory of representation of large-scale space based upon the observation and re-acquisition of distinctive visual events, i.e. landmarks. They are able to integrate the visual information from many images into a uniform database as a sensor-based robot moves through its environment. The representation provides the foundations for visual memory databases and path planning. They demonstrate their claims with a navigation simulator.

Talluri and Aggarwal [86–88] recently developed a system which is the closest to our approach in solving the localization problem. They take different views of the horizon and use them to search the underlying map for possible robot locations. In general several of these views constrain the possible locations to a small area. They show some results on simulated data based on a digital elevation map.

Thompson *et al.* presents a formalism in [89] within which the localization (“drop off problem”) can be studied. They discuss the approach which is taken by an expert human map users to deal with localization, and they propose a preliminary computational model of the process.

Reviewing the existing systems, most work was done regarding the localization based on specific landmarks. The focus of our research is to solve the localization problem using the panoramic horizon, given a topographic map and the orientation of the observer.

5.3 Glossary

Let us first define the following terms:

Localization: Localization is the process of establishing a match between a particular location in the environment and the corresponding location on a map [89].

Map: A map in our definition is a topographic map which provides us with the three dimensional coordinates for each surface point (e.g. range data). The z component (altitude or height) can be retrieved from the x and y coordinates: $z = \text{Map}(x, y)$. We assume that the map is small with respect to the associated planet sphere. Therefore the map can be considered as a small planar patch. The z axis is perpendicular to this patch.

Viewer or Observer: The viewer is an observing platform with a camera. In addition it can retrieve direction information based on a compass (this ability speeds up the localization significantly). The observer has the coordinates $v = (x_v, y_v, z_v)$. The camera is located above the surface: $z_v = \text{Map}(x_v, y_v) + h$, where h is the height of the camera.

Horizon: The horizon is the upper bound of the projection of all landscape points on a cylinder around the observer as shown in Figure 5.1. Such a panoramic horizon H is a periodic curve. Projecting the curve back on the landscape results in a set of three dimensional points \bar{H} whose projection on the map is shown in the results section. H and \bar{H} are defined in the following way (see Figure 5.2):

$$H(v) = \left\{ \Delta z(d)/d \mid \max_d \{ \Delta z(d)/d \}, \right. \\ \left. 0^\circ \leq \theta < 360^\circ \right\} \quad (5.1)$$

$$\bar{H}(v) = \left\{ p_v(\theta, d) \mid \max_d \{ \Delta z(d)/d \}, \right. \\ \left. 0^\circ \leq \theta < 360^\circ \right\} \quad (5.2)$$

with

$$\begin{aligned} v &= (x_v, y_v, z_v) \\ p_v(\theta, d) &= (x_v^\theta, y_v^\theta, z_v^\theta) \\ d &= \sqrt{(x_v^\theta - x_v)^2 + (y_v^\theta - y_v)^2} \\ \Delta z(d) &= z_v - z_v^\theta \end{aligned}$$

We call the tangent of α the *relative height* r , with $r(d) = \frac{\Delta z(d)}{d}$. Without loss of generality we set $0^\circ = 360^\circ$ as the north direction and define the orientation of a horizon as clockwise (as seen from above). An example can be seen in Figure 5.3. We display a horizon as a graph as seen from the observer. The abscissa represents the direction θ ($0^\circ = \text{north}$), the ordinate represents the relative height r . The graph of the horizon of Figure 5.3 can be seen in Figure 5.4.

It should be noted that a horizon can be occluded and changed by environmental influences such as fog or clouds. However, our algorithm does not rely

Figure 5.2: Slice Through a Landscape at an angle θ

Figure 5.3: Map with Superimposed Reprojected Horizon 1

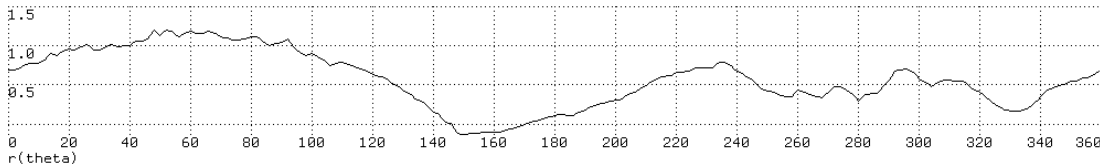
5.4 The Algorithm

5.4.1 The Basic Idea

As mentioned in [89] there is an important relationship between localization and recognition. Many of the computational tools that have proven useful for recognition turn out to be also relevant to localization. In outdoor navigation, the relevant “model” is a representation of the topographic features visible from a particular



(a) Panoramic View of Observer (rendered)



(b) Extracted Horizon Curve

Figure 5.4: Graph of Horizon 1

vantage point. The number of vantage points is effectively unbounded. So far however, the common belief is that this fact does not allow the use of a bounded set of models for the localization problem. This is why most approaches try to assemble the topographical information adaptively from the map. We show in this chapter that the localization problem can be reduced to a recognition problem when the system is able to deal with a data base which includes a large set of horizon models. Our localization approach is related to our early work [78], which addresses the problem of recognition of multiple flat objects in a cluttered environment from an arbitrary viewpoint. It allows to perform fast recognition even with large data bases. We exploit this fact to store a large set of horizons from a grid of vantage points and by using indexing we find the “most similar” horizon.

5.4.2 Representation of a Horizon

Offline Computation from the Map: A horizon is computed from a map by computing a set of sample slices at equal angles around the location v . Starting at 0° (north) a sample slice is computed through the landscape (see figure 5.2) every $\Delta\theta$ angle (typically $0.5^\circ \leq \Delta\theta \leq 3^\circ$). For every slice at the angle θ the point p_v^θ is obtained. From p_v^θ and v we compute the value for the tangent r_v^θ . The set of all points (θ, r_v^θ) with $0^\circ \leq \theta < 360^\circ$ represents the horizon.

Computation of the Observable Horizon: An observer located in an unknown environment can extract the panoramic horizon by

Figure 5.5: Two-Dimensional Horizon Super Segment

1. acquiring a panoramic image,
2. segmenting the “sky” from the “ground”,
3. linking the horizon curve,
4. and sampling the horizon curve.

In this thesis, we are not addressing the crucial segmentation step.

Encoding: Our encoding of a horizon is based on a polygonal approximation. We are not dependent on any feature detection algorithm and we do not have to handle explicit distinguished points like corners or inflection points. Our opinion is that curvature is the most important feature of a general curve. It is invariant with regard to scale, rotation and translation. By using a polygonal approximation we lose most of the curvature information, but we keep parts of this information in the angles of consecutive line segments.

Obviously, there is not a unique polygonal approximation for a horizon curve. Therefore, for the purpose of robustness, we use *several* polygonal approximations with different line fitting tolerances. Since we want to handle occlusion, we do not expect to obtain complete horizons, but only portions of them. On the other hand, individual segments are too local to be useful as matching primitives. Grouping a fixed number of adjacent segments provides us with our basic features, the super segments. In order to manipulate and to encode super segments we have to define some terms and to describe some attributes (see figure 5.5):

Cardinality The cardinality of a super segment is the number of segments it consists of.

Angles Let a super segment consists of n segments, then we have $n - 1$ angles between successive segments.

Direction A super segment represents a part of a horizon. The first vertex is located at the direction θ_{start} , the last vertex is located at θ_{end} . We define the direction of a super segment ss as the middle between the start and the end vertex:

$$\text{dir}(ss) = \begin{cases} \frac{\theta_{start} + \theta_{end}}{2} & \text{if } \theta_{start} < \theta_{end} \\ \frac{(\theta_{start} + \theta_{end} + 360) \bmod 360}{2} & \text{otherwise} \end{cases}$$

Relative Height Range The relative height $r(ss)$ of a super segment ss is defined as the middle between the minimal and the maximal relative height.

$$r(ss) = \frac{r_{\max}(ss) + r_{\min}(ss)}{2}.$$

The relative height range $\Delta r(ss)$ is defined as the difference between the maximum relative height and the minimum relative height:

$$\Delta r(ss) = r_{\max}(ss) - r_{\min}(ss).$$

As mentioned before, we are mainly interested in the curvature information implicitly captured by the super segment angles. This is the reason why we use them to encode a super segment. To avoid establishing matches between super segments which have the same angles but totally different ranges of relative height, we add the relative height Δr to our coding scheme. To encode a super segment ss with cardinality n we use a simple encoding scheme. The list of the quantized curvature angles and the relative height range is the code of the super segment ss :

$$\text{Code}(ss) = (\text{Quant}(\kappa_1), \text{Quant}(\kappa_2), \dots, \text{Quant}(\kappa_{n-1}), \text{Quant}(\Delta r))$$

All the encoded super segments serve as keys into a table (the data base), where we record the corresponding super segments as entries, as explained later.

In an early version of the system, we encoded in addition to the angles and the relative height the direction of the super segment. We saw a decrease in the performance of the system due to the fact that super segments which are close to the observer are relatively unstable in the direction with respect to little changes of the viewpoint. Therefore we had two options: the usage of a large quantization

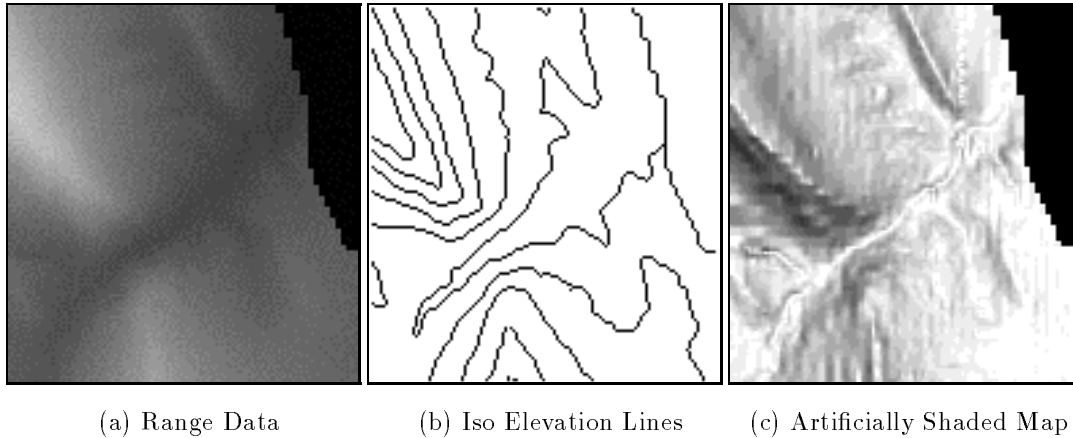


Figure 5.6: Different Representations of a Map

or the exclusion of the direction information from the code. The usage of a large quantization did not improve the results of the matching significantly, therefore we chose the second way and use the direction information not in the hypotheses retrieval but in the verification step (see section 5.4.5).

5.4.3 Representation of the Map

A map can be visualized in several ways. In figure 5.6 we show in (a) the intensity coded range data of a map, in (b) the isoelevation lines, known from topographic maps, and in (c) the artificially shaded range data under the assumption that the surface obeys Lambertian reflectance properties. Our computations are done on the range data, however, for better visibility of our figures we display the shaded data.

We represent the map by a set of horizons computed for the points of a grid superimposed on the map (see figure 5.7). The spacing of the grid determines the accuracy with which we can find the correct location. To find the correct horizon which corresponds to the horizon observed by the observer, we want the encoding to be compact, accessible fast, and the storage of a large number of horizons should be possible. We choose for these reasons a data structure which is implemented as a hash table. A hash table allows efficient storage (only pointers are recorded), the hashing scheme allows fast access and different super segments with the same keys can be stored in cellar like buckets. The encoding of a map consists of the following steps (see figure 5.8):

1. Compute the horizons H_k of the map.

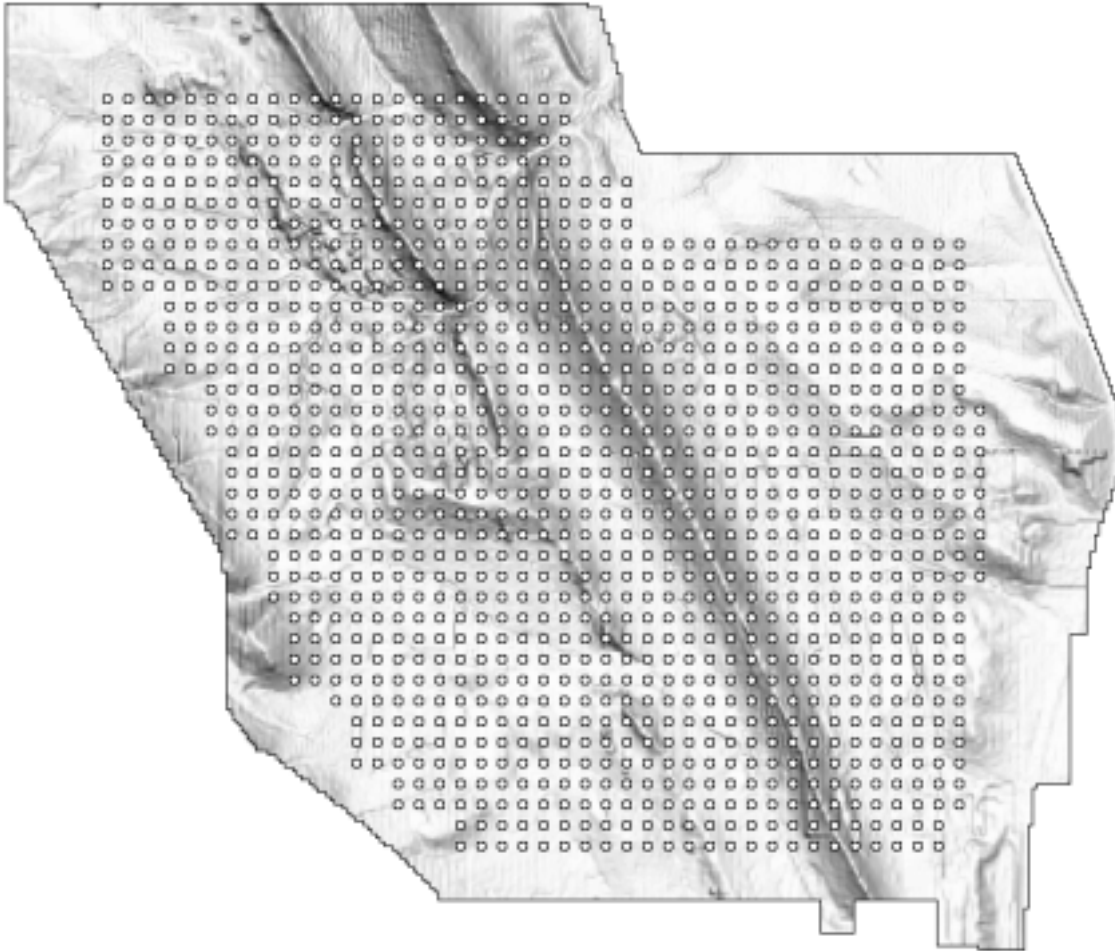


Figure 5.7: Map with Superimposed Horizon Grid

2. Compute the super segments $^{k}ss_i$ of H_k .
3. Encode the super segments. For every super segment several codes Code_j are computed. They are related to the different line fitting tolerances. Every code of every super segment serves as a key for an entry in a table (the data base) where we record the the super segment and the corresponding horizon.

The table (data base) grows in size with the number of recorded horizons. This process of building the data base from the map is performed off line.

Figure 5.9: Hypotheses Generation

of the following steps:

1. The horizon is preprocessed to generate all the super segments as explained in section 5.4.2.
2. The super segments are encoded.

Figure 5.10: Hypothesis Errors

3. The encoded super segments are used to retrieve the candidate hypotheses between the super segments of the visible horizon and the super segments of the horizons of the map.

As described in section 2.5, the quantization size is crucial for the encoding of the features. Typical values are (only for the quantization of the angles):

cardinality	3	4	5	6	7	8	...
interval size	20	30	40	45	60	60	90

5.4.5 Hypotheses Verification

The task of the verification is to distinguish *good* from *bad* hypotheses. Good hypotheses correspond to true matches, bad hypotheses correspond to wrong matches. To talk about hypotheses we have to define the following terms (see also figure 10).

- A hypothesis h consists of two super segments: ss_m is the super segment of the map horizon, ss is the super segment of the observed horizon ($h = \langle ss_m, ss \rangle$).
- The cardinality of a hypothesis h is defined as: $c(h) = \text{cardinality}(ss_m) = \text{cardinality}(ss)$.
- The direction error is defined as: $\varepsilon_d(h) = \|\text{dir}(ss_m) - \text{dir}(ss)\|$.

Figure 5.11: Hypotheses Verification

- The angle error is defined as: $\varepsilon_a(h) = \frac{1}{c-1} \sum_{i=1}^{c-1} \|\kappa_i^m - \kappa_i\|$ with $c = \text{cardinality}(ss)$.
- The vertex error is defined as: $\varepsilon_v(h) = \frac{1}{c} \sum_{i=1}^c \|\Delta\theta_i\|$.

Good hypotheses consist of similar super segments with similar directions and therefore $\varepsilon_d(h) \approx 0$, $\varepsilon_a(h) \approx 0$, and $\varepsilon_v(h) \approx 0$.

Geometrical Constraints: If more than one hypothesis votes for a specific horizon, we have to verify the consistency of the geometrical constraints. Two hypotheses 1h and 2h are consistent when

1. $\|\text{dir}({}^1ss) - \text{dir}({}^2ss)\| \approx \|\text{dir}({}^1ss_m) - \text{dir}({}^2ss_m)\|$ with ${}^i h = \langle {}^i ss_m, {}^i ss \rangle$
2. and $\|\text{r}({}^1ss) - \text{r}({}^2ss)\| \approx \|\text{r}({}^1ss_m) - \text{r}({}^2ss_m)\|$.

Subsumption: A super segment ss_i is subsumed by another super segment ss_j if the vertices of ss_i are a subset of the vertices of ss_j . As an extension we say a hypothesis 1h is subsumed by another hypothesis 2h when their super segments are subsumed by each other: 1ss_m is subsumed by 2ss_m , and 1ss is subsumed by 2ss . When one hypothesis is subsumed by another hypothesis we remove the hypothesis with the smaller cardinality.

Therefore, the verification stage consists of the following steps (see figure 5.4.5).

1. We compute all possible matches for the observed horizon with the map horizons to generate the hypotheses.

2. We filter out the hypotheses which are subsumed by other hypotheses. We also remove hypotheses which have large errors for ε_d , ε_a , or ε_v (typically we require that $\varepsilon_d, \varepsilon_a, \varepsilon_v < 20^\circ$).
3. After removing the hypotheses which are unlikely to represent a good match, we want to find the best hypothesis of the remaining ones. This is done using a simple heuristic. We can sort the hypotheses based on the values of ε_d , ε_a , ε_v , and the cardinality. We are interested in the largest possible match (maximal cardinality) with the minimal error (minimal $\varepsilon_d, \varepsilon_a, \varepsilon_v$).

$$w(h) = w_d \varepsilon_d(h) + w_a \varepsilon_a(h) + w_v \varepsilon_v(h) + w_c c(h)$$

Typically we use $w_d = w_a = w_v = -1$ and $w_c = 2$ to punish errors and to reward high cardinality. This results in an ordered set of hypotheses. We assume that the best hypothesis represents the best match. Until this point we did not consider the case that more than one hypothesis can vote for the same map horizon $H(v_{best})$. Using the above mentioned geometrical constraints described above, we check the consistency of the best hypothesis with the rest of the hypotheses in order to find other hypotheses which vote for the same map horizon $H(v_{best})$. The location of the map horizon v_{best} of these consistent hypotheses is the wanted location, where the horizon line resembles most the observed horizon as seen from v .

This mechanism works well as we show in the next section.

5.5 Results

5.5.1 Results on Simulated Data

The localization algorithm is now illustrated with data examples from a real terrain map. The horizons which are seen by the observer are simulated. For the presentation of the range data we always display the artificially shaded images.

As a terrain map we use the DEM (digital elevation model) covering the Martin Marietta ALV test area. A digital elevation model is a two-dimensional array of uniformly spaced terrain elevation measurements. Our DEM map consists of 810 pixel \times 702 pixel with a pixel corresponding to a size of 5×5 m² on the ground. The whole map corresponds to an area of approximately 4×3.5 km². Outside of the map we assume flat surface (sometimes the horizon line is not limited to the area described by the map). The horizons on the map are sampled on a grid with a grid spacing of 15 pixels. This corresponds to 75 m. For better visibility we exaggerate

the elevation data. The lowest elevation is 21 (in pixels), the highest is 252. The height of the observer above the surface was always constant (2 pixels). For the linear approximation we used the line fitting tolerances 2, 3, 4, and 5. An example is shown in figure 5.12(a) to (d). For the matching we used super segments of cardinality 6 and 8. To obtain useful linear approximations, we scaled the relative height by a factor of 50. These are no critical values, as significant deviations from these values do not affect the results. We choose three examples to illustrate different aspects of our system:

1. Horizon 1 is a general horizon with no specific difficulties.
2. In horizon 2, the observer is located close to a steep ridge.
3. For horizon 3 the system does not find the optimal location.

The shaded map with the super imposed grid of 1211 horizons is shown in figure 5.7. The localization in all three examples took less than 1 minute.

Horizon 1: In the example of horizon 1, the observer is located in a valley between two ridges (see figure 5.3). figure 5.4 shows the rendered panoramic view from the location of the observer (a) and the extracted horizon curve (b). After the matching the system finds the hypotheses as shown in figure 5.13. In figure 5.14 we display the detected corresponding super segments of the best hypothesis. The overlaid numbers correspond to the angles of the super segments. This hypothesis was found among the hypotheses shown in figure 5.15(a) and can be seen in a closeup in figure 5.15(b). figure 5.16 shows the rendered view seen by the observer in comparison with the rendered panoramic view of the detected result.

Horizon 2: Horizon 2 is the horizon seen from the point in figure 5.17. The observer is located very close to a ridge. The horizon curve is shown in figure 5.18. For this localization example, we do not show all the possible hypotheses, because the correct hypothesis was the only one which “survived” the filter process. A closeup view of the final result is shown in figure 5.20. The corresponding rendered views are displayed in figure 5.21.

Horizon 3: In the example of horizon 3 the observer is located on a slope with a wide view (see figure 5.22). The horizon graph is shown in figure 5.23. The hypothesis in figure 5.24 is the best of the hypotheses in figure 5.25(a). As shown in figure 5.25(b), this is only the third closest location. This is due to the fact that the observer is close to the border of the map and therefore only approximately only 180°

of horizon are useful for matching. Furthermore, the significant horizon part does not constrain the distance enough, so that the hypotheses have one degree of freedom, and are approximately clustered along a line, pointing towards the significant horizon part (see figure 5.25(a)). This behavior can be improved by changing the heuristic of finding the best hypothesis, but it can not be excluded in general. The corresponding rendered views are displayed in figure 5.26.

Changing the Height of the Observer We performed some experiments by changing the height of the observer above the surface. Small changes (up to a height of 10 pixels) have no influence on the recognition performance. Increased height (30 and more) adds an increased uncertainty to the localization result. The results show one degree of freedom, and are distributed along a line pointing in the direction of the most stable features (the part of the horizon which is furthest away).

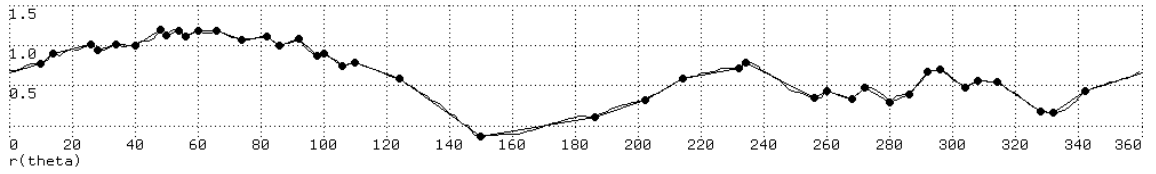
5.5.2 Results on Real Data

In contrast to the last section, we now show an example which is based on a real horizon (see figure 5.27(a)) taken in the Wasatch National Forest, east of Salt Lake City². The panoramic view was obtained by registering the digitized images manually. For every two adjacent images, we picked corresponding points in the overlapping area, and with a least squares method we computed the transformation. The crest line (see figure 5.27(b)) was computed by thresholding the image.

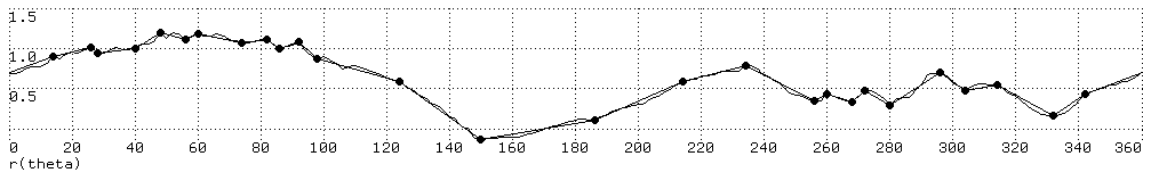
The terrain map is the DEM of the surrounding area (see figure 5.28). The map consists of 709 pixel \times 468 pixel with a pixel corresponding to a size of 30 \times 30 m² on the ground. The horizons on the map are sampled on a grid with a grid spacing of 10 pixels. This corresponds to 300 m. We compute approximately 1500 horizons. For the linear approximation we use the line fitting tolerances 4, 5, and 6. For the matching we use super segments of cardinality 5, 7 and 9. To obtain useful linear approximations, we scale the relative height of the DEM by a factor of 10.

Due to lack of a camera model we are not able to match the horizon curve directly with the DEM. We have one degree of freedom which is the relative height. We decided to overcome this problem by using several scales for the horizon curve simultaneously. The localization process takes less than one minute and results in the detection shown in figure 5.29. In figure 5.30 we display the detected corresponding super segments of the best hypothesis. Figure 5.31 shows the comparison between the view of the observer and the rendered view as seen from the computed location.

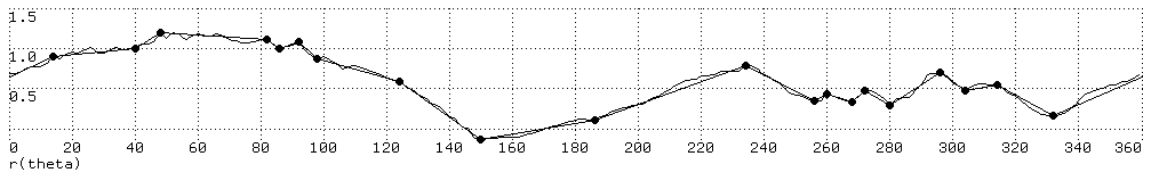
²Special thanks to Thomas Colvin and Dr. William Thompson from the University of Utah who made the slides and the DEM available.



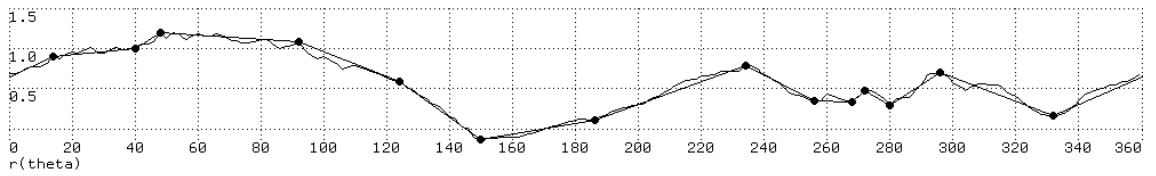
(a) Linear Approximation with Fitting Tolerance 2



(b) Linear Approximation with Fitting Tolerance 3



(c) Linear Approximation with Fitting Tolerance 4



(d) Linear Approximation with Fitting Tolerance 5

Figure 5.12: Approximations of Horizon 1

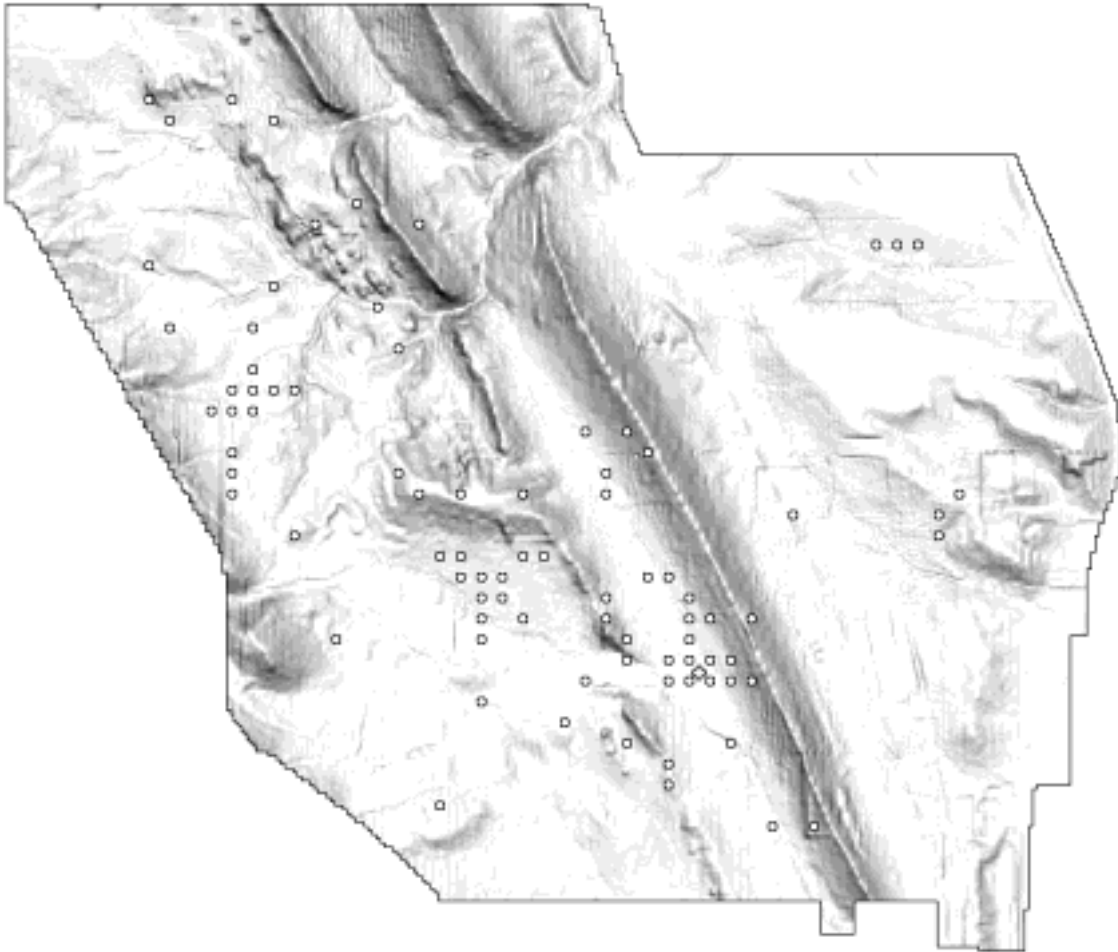
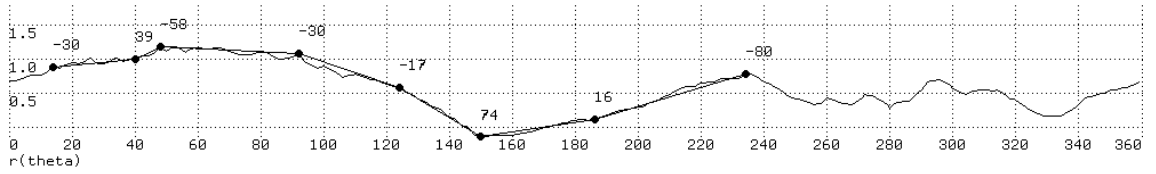
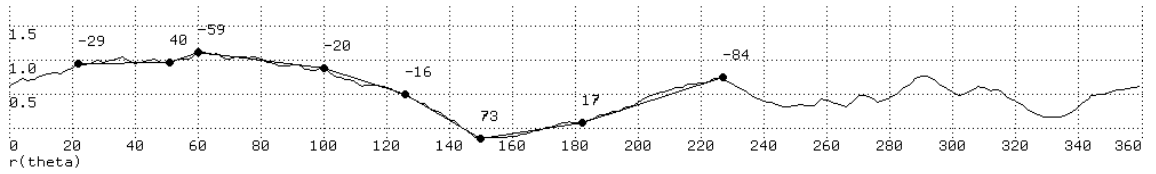


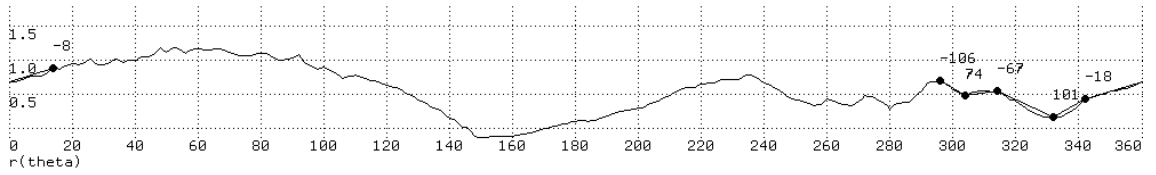
Figure 5.13: Hypotheses after Retrieval



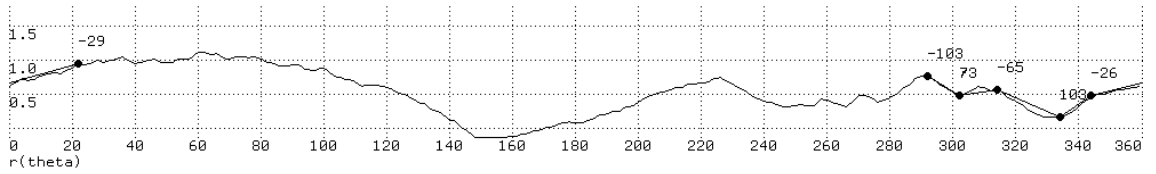
(a) Matched Super Segment #1 of Observer



(b) Corresponding Super Segment of Map

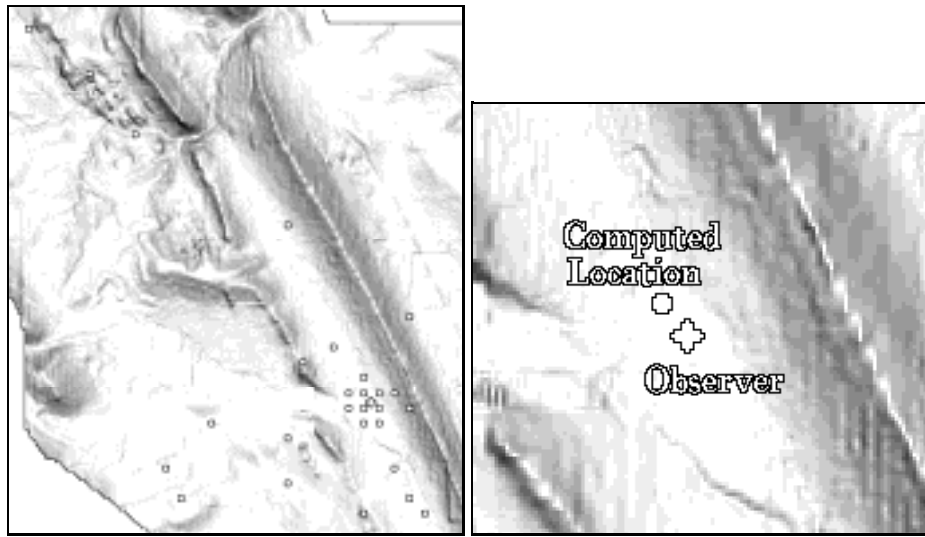


(c) Matched Super Segment #2 of Observer



(d) Corresponding Super Segment of Map

Figure 5.14: Hypotheses for Horizon 1



(a) Hypotheses after Filtering

(b) Closeup of the Detected Result

Figure 5.15: Result 1



(a) Rendered Panoramic View of Observer



(b) Rendered Panoramic View of Computed Location

Figure 5.16: Simulated Panoramic Views for Horizon 1

Figure 5.17: Map with Superimposed Reprojected Horizon 2

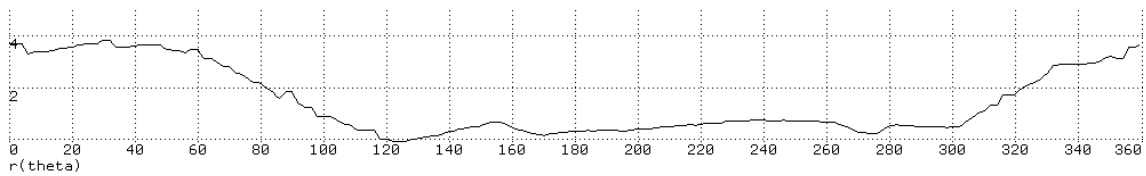
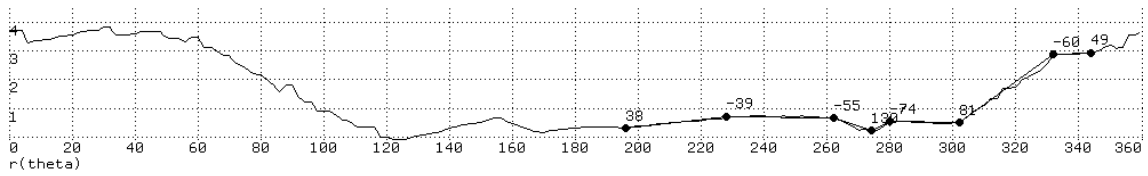
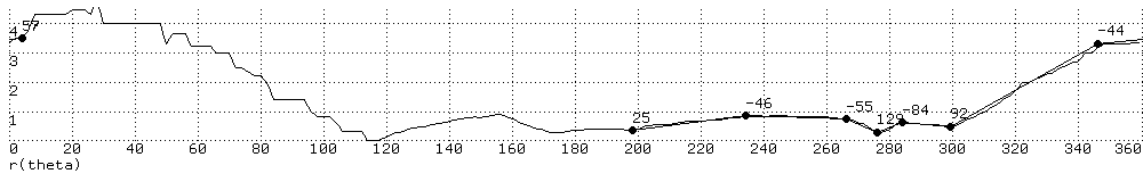


Figure 5.18: Graph of Horizon 2



(a) Matched Super Segment of Observer

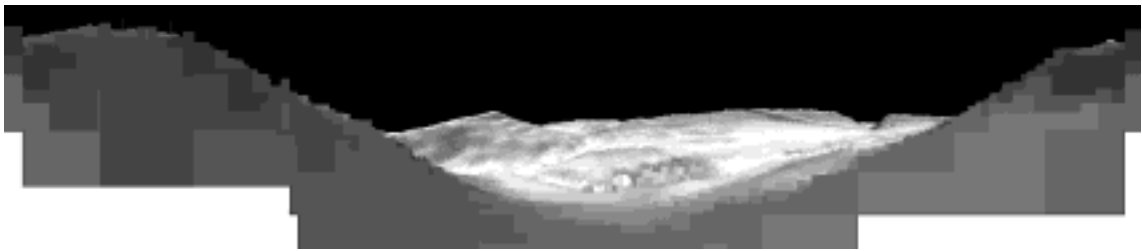


(b) Corresponding Super Segment of Map

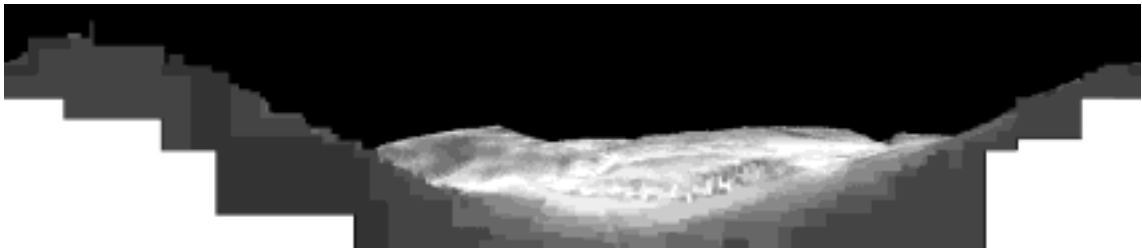
Figure 5.19: Hypothesis for Horizon 2



Figure 5.20: Result 2



(a) Rendered Panoramic View of Observer



(b) Rendered Panoramic View of Computed Location

Figure 5.21: Simulated Panoramic Views for Horizon 2

Figure 5.22: Map with Superimposed Reprojected Horizon 3

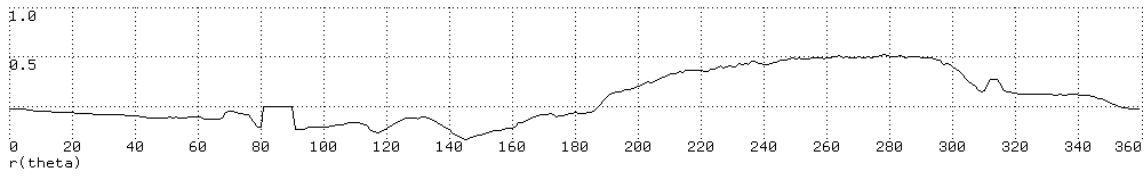
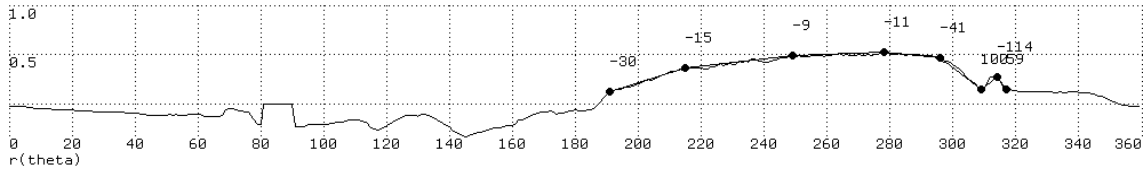
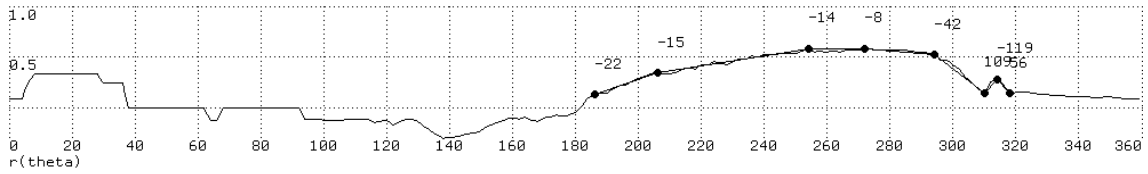


Figure 5.23: Graph of Horizon 3

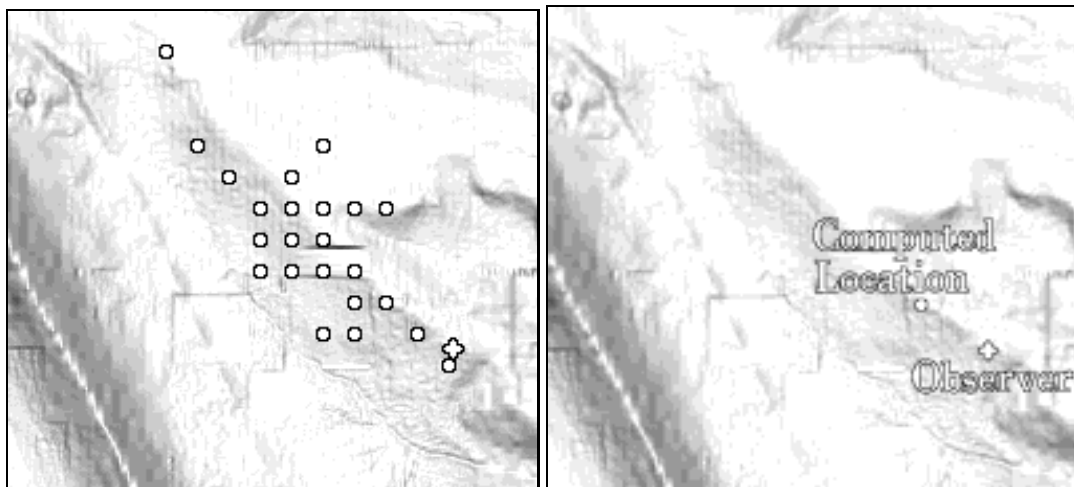


(a) Matched Super Segment of Observer



(b) Corresponding Super Segment of Map

Figure 5.24: Hypothesis for Horizon 3



(a) Hypotheses

(b) Detected Result

Figure 5.25: Result 3



(a) Rendered Panoramic View of Observer



(b) Rendered Panoramic View of Computed Location

Figure 5.26: Simulated Panoramic Views for Horizon 3

(b) Crest Line

Figure 5.27: View from Albion Campground

(b) Horizon

Figure 5.29: Result

(b) Corresponding Super Segment of Map

Figure 5.30: Hypothesis for Result

Figure 5.31: Panoramic and Reconstructed View from Albion Campground

5.6 Conclusion

We have presented an algorithm which is able to perform the localization based on the panoramic horizon. We have demonstrated the performance on simulated and real data.

From our experience the algorithm works well. However, the system was not able to perform the correct localization when we put the observer in a small valley where the horizon line is extremely unstable and changes occur caused by minimal alteration of the viewpoint. The general rule is that the further the view is, the more likely it is that the system finds the correct location. To counteract this behavior, an improvement of the sampling of the map could be performed by sampling areas with unstable horizons with a smaller grid spacing than areas with a stable horizon. This would improve the performance significantly.

Another issue is the size of the map. Our system fails to recognize the correct horizon, when features (like a high mountain) from outside the map are part of the horizon curve visible by the observer. Therefore we have to use a map which covers a sufficient large area, to include all landmarks, which are visible from the area in which we want to perform the localization task.

Chapter 6

Conclusion and Future Research

6.1 Summary

In this dissertation we have introduced a mechanism called *structural indexing*. The basic idea is that we store structural primitives of a set of given models into a table. The recognition proceeds by collecting the structural primitives of a scene and retrieving all the corresponding elements from the table in order to form matching hypotheses. Finally a clustering of the hypotheses is performed with respect to a set of constraints which assigns every hypothesis a membership to a model.

We have addressed the design of the structural primitives in the following domains:

- For the recognition of flat objects we use planar super segments as structural tokens. They are computed from edge data.
- For the recognition of three-dimensional objects from range images we use three-dimensional super segments as matching primitives, which are either derived from three-dimensional edges or from radial decompositions, called *splashes*, which represent small surface patches.
- For the recognition of three-dimensional models from two-dimensional grey level images we use subgraph matching. The graphs consist of primitives which are computed from perceptual groups. Small subgraphs provide the structural tokens.

The design of the underlying features defines both the strengths and the limitations of the approach. When the assumptions about the data hold true, our algorithms are efficient, they can handle large data bases, they are redundant to noise, and they can cope with a large variety of objects. We illustrated this with several examples.

The limitations are directly connected to the breakdown of the primitive extraction. We have discussed the issues for the recognition of flat objects and for

the recognition of three-dimensional objects from range images, where we experience serious difficulties when primitives cannot be computed due to the break of co-curvilinearity (dense data assumption). The limitations become really evident in our approach of recognizing three-dimensional models from grey level scene data. We deal with this by using, in addition to co-curvilinearity, other grouping strategies such as parallelism, symmetry and closure. We have introduced a novel method to compute these groups in an efficient way and to construct graphs which can be matched using structural indexing. This approach is not a mature system such as the algorithm dealing with flat objects or the TOSS system, but we have demonstrated that it is a big step in the right direction and that the idea holds promise.

6.2 Future Research

Looking into the future is difficult, but we can give some directions which we believe are crucial for future contributions in the field of object recognition. Designing powerful features is one of the focus points. The idea of the super segment, the splash, and the combination of convex approximations is just a beginning. Future systems in object recognition have to further exploit the fact that *rich* features provide enough structural information to recognize objects efficiently, even from a large data base.

We strongly believe that the following two issues will play an important role. They represent the two directions of feature design: lower, more local, and higher, more global:

- What happens when the system does not find corresponding high level groupings (e.g. due to heavy occlusion)? A solution could lie in a multilevel matching scheme, which allows the system to “fall back” on lower level features in order to find correspondences.
- We stopped the grouping process in our algorithms at a certain stage. An extension would be the organization of high level features to even higher levels. This would result in extremely rich descriptions which would finally result in a description where we can talk of *parts* of an object as features.
- There are several other feature and grouping strategies which we ignore so far: continuation, texture, saliency ... Including these features would enrich the descriptive and discriminative power of our algorithms.

Appendix A

Least Square Method

In this section we introduce a least square method for finding the transformation matrix between two sets of three dimensional points by minimizing the error. The correspondences are known. The method is based on the two dimensional approach in [48, 49]. The question is: what is the best match between the sequences of points $(u_j)_{j=1}^n$ and $(v_j)_{j=1}^n$. The goal is to find the transformation $T(u) = Au + b$ which minimizes the distance between the sequences $(Tu_j)_{j=1}^n$ and $(v_j)_{j=1}^n$: $\delta = \min_T \sum_{j=1}^n |Tu_j - v_j|^2$. To later eliminate a second order term, translate the set (u_j) so that $\sum_{j=1}^n u_j = 0$. Then

$$\delta = \min_{A,b} \sum_{j=1}^n |Au_j + b - v_j|^2 =$$
$$\min_{A,b} \left(\sum_{j=1}^n |b - v_j|^2 + \sum_{j=1}^n |Au_j|^2 + 2 \sum_{j=1}^n b \cdot Au_j - 2 \sum_{j=1}^n Au_j \cdot v_j \right).$$

With

$$\sum_{j=1}^n b \cdot Au_j = b \cdot A \left(\sum_{j=1}^n u_j \right) = 0.$$

we can eliminate one second order term and therefore we can minimize $\sum_{j=1}^n |b - v_j|^2$ and $\sum_{j=1}^n |Au_j|^2 - 2 \sum_{j=1}^n Au_j \cdot v_j$ separately. The solution for b is simply

$$b = \frac{1}{n} \sum_{j=1}^n v_j.$$

To minimize over A we set

$$g(A) = g(a_{11}, a_{12}, \dots, a_{33}) = \sum_{j=1}^n |Au_j|^2 - 2 \sum_{j=1}^n Au_j \cdot v_j.$$

We have to find

$$\min_A g(A) = \min_A g(a_{11}, a_{12}, \dots, a_{33})$$

by solving the following system of 9 equations ($i = 1, 2, 3; j = 1, 2, 3$):

$$\frac{\partial g}{\partial a_{ij}} = 0 \tag{A.1}$$

Since g is a quadratic function in each of its unknowns, (A.1) is a system of nine linear equations with nine unknowns; the nine equations are three independent sets of equations with three unknowns. The computation of the unknowns (the transformation matrix T) is straightforward.

The determinant of the rotational part A of the transformation matrix provides us with a measurement for the volume change imposed by the transformation. The transformation is rigid if $\det(A) = 1$. When we examine a hypothesis on rigidity we have to consider the effect of noise (see Section 3.5.3). Therefore, in practice, we validate the rigidity assumption only if $0.5 < \det(A) < 2.0$.

Appendix B

Consistency between Hypotheses

We simplify the presentation by regarding a hypothesis in this section as a match between two splashes whereas we actually also have to consider matches between 3D super segments. The extension to a hypothesis based on 3D super segments is straightforward and is addressed at the end of every subsection. It is important to note that, with a match of two features, we get a set of corresponding points. For example:

- A hypothesis of two splashes of cardinality four provides us with five corresponding point pairs. Four are in the vicinity of the patch (along the sample normals). They correspond to the vertices of the polygonal approximation in the (θ, ϕ, ψ) space. The fifth pair corresponds to the centers of the two splashes (the locations).
- A hypothesis of two 3D super segments of cardinality four provides us with five corresponding point pairs, one for each vertex.

Suppose we have two hypotheses h_1 and h_2 : Every h_i consists of a match between a splash on the model m_i and a splash on the scene s_i . The reference normal of a splash s is n_s . Now we address the question: when are the two hypotheses h_1 and h_2 mutually consistent? We basically have to reduce the five degrees of freedom (*dofs*) which describe the geometrical relationship between our two features to completely constrain the attitude of one feature relative to another. We have to deal with three *dofs* for the relative position, and two *dofs* for the relative orientation.

B.0.1 The Distance Constraint

- The distance between two splashes is the distance between their locations. To be consistent, the two splashes on the model should have approximately the same distance as the splashes on the scene (since it is a rigid transformation) as shown in Figure B.1(a).

(b) Orientation
Constraint

Figure B.1:

- Therefore we require: $\|dist_m - dist_s\| < \varepsilon_d$.
- With this constraint we reduce the dofs for the position to two.
- *3D super segments*: For 3D super segments we use the middle vertex (or in case of even cardinality the middle point of the middle segment) as the feature position.
- In our implementation we use $\varepsilon_d = 0.1 \cdot \max(dist_m, dist_s)$.

B.0.2 The Orientation Constraint

- For a splash we use the reference normal as the orientation vector. The angle α_m which is formed by the orientation vectors of the two splashes on the model should be approximately the same as the angle α_s which is formed by the orientation vectors of the two splashes on the scene (Figure B.1(b)).
- Therefore we require: $\|\alpha_m - \alpha_s\| < \varepsilon_\alpha$
- This reduces the orientation dofs to one.
- *3D super segments*: We use the vector from the last to the first vector as the orientation vector for a 3D super segment.
- In our implementation we use $\varepsilon_\alpha = 25^\circ$.

(c) Assigning a Frame to a 3D Super Segment

Figure B.2: Direction Constraint

B.0.3 The Direction Constraint

- For computing the direction angles γ_{1i} and γ_{2i} for splash s_2 in relation to s_1 we have to assign a unique frame F to the splash s_1 , as shown in Figure B.2 (this frame should not be confused with the frame we defined in Section 3.3.2).
 - The frame F has its origin at the location of the splash.
 - The z axis is the reference normal.
 - In Section 3.3.2 we use the point with the strongest tangent tilt as the start of the polygonal approximation. This point p is the point where the sample normals have the strongest tilt with respect to the reference normal and we know that p lies on the circular slice of the splash. We know further that p has a corresponding point on the matched splash,

Without loss of generality we use p as a reference point for defining the x axis.

- The x axis is defined as the vector which is perpendicular to z and lies in the plane defined by z and the point p .
- The y axis is perpendicular to x and z (and x, y, z form a right handed coordinate system).

Representing the vector $s_1 \vec{s}_2$ in spherical coordinates in frame F results in an angle pair $(\gamma_1^{s_1}, \gamma_2^{s_1})$ (see Figure B.2(d)).

To preserve the rigidity assumption, the values for $(\gamma_1^{s_1}, \gamma_2^{s_1})$ must be approximately the same as for $(\gamma_1^{m_1}, \gamma_2^{m_1})$. This reduces the dofs for the position by two, to zero. Requiring also similar values for $(\gamma_1^{s_2}, \gamma_2^{s_2})$ and $(\gamma_1^{m_2}, \gamma_2^{m_2})$ reduces the dofs for the orientation to zero.

- Therefore we require for $i, j \in \{1, 2\}$: $\|\gamma_i^{s_j} - \gamma_i^{m_j}\| < \varepsilon_\gamma$
- *3D super segments*: For a 3D super segments we assign a frame F in the following way (see Figure B.2(e)):
 - The frame F has its origin at the first vertex.
 - The z axis is the (normalized) orientation vector.
 - The x axis is perpendicular to z and lies in the same plane as the first segment. If the first segment and the orientation vector are aligned, take the next segment.
 - The y axis is perpendicular to x and z in a right handed coordinate system.
- In our implementation we use $\varepsilon_\gamma = 25^\circ$.

Appendix C

The Hashing Scheme

A classical problem in computer science is how to store information dynamically to allow for quick access. This is a basic searching problem which is widely addressed in the literature. Each package of information is stored as a record. Each record consists of a key that uniquely identifies it. The task of a searching algorithm is to take an input K and return the record (if any) that has K as its key. One important solution to this problem is provided by *hashing*, because no matter how many records are stored, the average search times remain bounded. Suppose we want to store N records in a contiguous area of memory containing at least M locations. The common element of all hashing algorithms is a pre-defined hash function

$$\text{hash} : \{\text{all possible keys}\} \rightarrow \{1, 2, \dots, M\}$$

that assigns the N records to M memory slots in a uniform manner. Hashing algorithms differ from one another in how they resolve the collision that results when the memory location assigned to a record is already occupied. We do not want to go in further detail, but invite the interested reader to consult references and motivation in the excellent book written by Jeffrey S. Vitter and W. Chen with the title *Design and Analysis of Coalesced Hashing* [96].

In our algorithm we use several tables. We need a table for the data base of the representation of the objects (Section 3.5.1), we need tables for the correspondences and the clusters (Section 3.5.3). All these tables should allow fast access. Because our algorithm is implemented in Symbolics Common Lisp, we use the hash table scheme offered by this language. It offers several interesting features:

- The internal representation changes at run-time. The most efficient representation is picked for the data currently stored in the table. The representation changes when the size of the data passes some size threshold, either upward or downward. Smaller tables are represented as association list, larger tables in a standard hashing scheme.

- The hash table can grow when it is filled up to a certain threshold. In this case the table performs automatic rehashing. The hash function is recomputed to optimally suit the stored data for minimal occurrence of collisions.

A noteworthy issue for our algorithm is the initial size for a table.

1. If we use a small initial table size, rehashing occurs every time the table grows. This process includes a continuous updating of the hash function, which minimizes collisions. On one hand the table buildup requires more time, on the other hand, the access will be faster (e.g. for matching) because the number of collisions is minimal.
2. If we set the initial size to a large value, the buildup of the table is faster, but several collisions occur and the retrieval of data is slower.

We prefer for the creation of our tables the first option to later have minimal retrieval times for e.g. fast matching.

Other approaches in object recognition which use some kind of hash table based indexing approach can be found in [11, 15, 31, 48, 49, 42]

(b)

Figure D.1:

and

$$0 \leq (\alpha_2 - \beta_2) \leq 2\theta. \quad (\text{D.4})$$

First let us focus on equation (D.3). We know from Figure D.1(b) that

$$\alpha_1 + \pi/2 + \theta + \gamma_1 = \pi \quad (\text{D.5})$$

and

$$\beta_1 + \pi/2 - \theta + \gamma_2 = \pi. \quad (\text{D.6})$$

Subtracting (D.5) from (D.6) results in

$$(\beta_1 - \alpha_1) + (\gamma_2 - \gamma_1) = 2\theta. \quad (\text{D.7})$$

Furthermore we know that

$$\gamma_2 + \gamma_1 \leq \pi. \quad (\text{D.8})$$

By showing that $\gamma_2 \geq \gamma_1$ we can show that (D.3) is true. The proof of $\gamma_2 \geq \gamma_1$ is divided into two cases:

1. $\gamma_2 \geq \pi/2$: Considering equation (D.8) we get $\gamma_2 \geq \gamma_1$.
2. $\gamma_2 < \pi/2$: Based on projective geometry we get

$$\gamma_2 \geq \gamma_1 \Leftrightarrow m_2 \geq m_1$$

with $m_2 = m_1 + c$. If we can show that $c \geq 0$, we show that $\gamma_2 \geq \gamma_1$. We know that

$$c = b \tan \theta. \quad (\text{D.9})$$

Furthermore we get from the projective geometry a relation between a and b :

$$a/x = b/2x \Leftrightarrow b = 2a. \quad (\text{D.10})$$

Since $a = x \sin \theta$ we get from equation (D.9) and (D.10) $c = 2x \sin \theta \tan \theta$, and therefore $c \geq 0$ for $0 \leq \theta \leq \pi/2$.

Therefore equation (D.3) is true. Equation (D.4) can be shown in the same way.

Q.E.D.

(b)

Figure E.1: Adjacency between CAs

two CAs as being adjacent when the area of the intersecting bounding boxes of the two segments is larger than 30% of the area of the union of both bounding boxes. The size of a bounding box is dependent on the length of the corresponding segment. The width is defined as being a fraction of the length. In our implementation we use 20%.

Figure E.2: Inclusion of CAs

Reference List

- [1] Y. Alavi, G. Chartrand, L. Lesniak, D. R. Lick, and C. E. Wall. *Graph Theory with Applications to Algorithms and Computer Science*. John Wiley and Sons, 1985.
- [2] R. C. Arkin, E. M. Riseman, and A. R. Hanson. Aura: And architecture for vision-based robot navigation. In *Proceedings of the DARPA Image Understanding Workshop*, pages 417–431, Los Altos, 1987.
- [3] N. Ayache and O. Faugeras. HYPER: A New Approach for the Recognition and Positioning of Two-Dimensional Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(1):44–54, 1986.
- [4] A. Beinglass and H. J. Wolfson. Articulated object recognition, or: How to generalize the generalized hough transform. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, Maui, Hawaii, June 1991.
- [5] A. Berztiss. A backtrack procedure for isomorphism of directed graphs. *Journal of the ACM*, 20(3):365–377, July 1973.
- [6] P. J. Besl. *Machine Vision for Three Dimensional Scenes*, chapter The Free-Form Surface Matching Problem, pages 25–71. Academic Press, 1990. Freeman, H. editor.
- [7] B. Bhanu. Representation and Shape Matching of 3-D Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(3):340–350, 1984.
- [8] T.O. Binford. Inferring surfaces from images. *Artificial Intelligence*, 17:205–244, 1981.
- [9] R. C. Bolles and R. A. Cain. Recognizing and locating partially visible objects: The Local-Feature-Focus method. *International Journal of Robotics Research*, 1(3):57–82, 1982.
- [10] R. C. Bolles and P. Horaud. 3DPO: A Three-Dimensional Part Orientation System. *International Journal of Robotics Research*, 5(3):3–26, 1986.
- [11] T.M. Breuel. Adaptive model base indexing. In *Proceedings of the DARPA Image Understanding Workshop*, pages 805–814, 1989.

- [12] R. A. Brooks. Symbolic reasoning among 3-D models and 2-D images. *Artificial Intelligence*, 17:285–348, 1981.
- [13] R. A. Brooks. Model-based three dimensional interpretations of two dimensional images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2):140–150, 1983.
- [14] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, pages 14–23, April 1986.
- [15] A. Califano and R. Mohan. Multidimensional indexing for recognizing visual shapes. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, pages 28–34, Maui, Hawaii, June 1991.
- [16] J. F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.
- [17] T. A. Cass. A Robust Parallel Implementation of 2D Model-Based Recognition. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, pages 879–884, Ann Arbor, Michigan, June 1988.
- [18] C. H. Chen and A. C. Kak. A Robot Vision System for Recognizing 3-D Objects in Low-Order Polynomial Time. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(6):1535–1563, 1989.
- [19] J.-S. Chen, G. Medioni, and A. Huertas. Fast convolution with Laplacian-of-Gaussian masks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(4):584–590, July 1987.
- [20] D. T. Clemens and D. W. Jacobs. Space and time bounds on indexing 3-d models from 2-d images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1991.
- [21] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
- [22] D. Corneil and C. Godtlieb. An efficient algorithm for graph isomorphism. *Journal of the ACM*, 17(1):51–64, January 1970.
- [23] M. Dhome, M. Richetin, J. T. Lapresté, and G. Rives. Determination of the Attitude of 3-D Objects from a Single Perspective View. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(12):1265–1278, 1989.
- [24] S. J. Dickinson, A. P. Pentland, and A. Rosenfeld. 3-d shape recovery using distributed aspect matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1992.

- [25] G. J. Ettinger. Large Hierarchical Object Recognition Using Libraries of Parameterized Model Sub-parts. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, Ann Arbor, Michigan, June 1988.
- [26] T. J. Fan. *Describing and Recognizing 3-D Objects Using Surface Properties*. Springer Verlag, New York, 1990.
- [27] T. J. Fan, G. Medioni, and R. Nevatia. Recognizing 3-D Objects Using Surface Descriptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(11):1140–1157, 1989.
- [28] O. Faugeras and Hebert M. The Representation, Recognition, and Locating of 3-D Objects. *International Journal of Robotics Research*, 5(3):27–52, 1986.
- [29] P. J. Flynn and A. K. Jain. On reliable curvature estimation. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, pages 110–116, San Diego, California, June 1989.
- [30] P. J. Flynn and A. K. Jain. 3d object recognition using invariant feature indexing of interpretation tables. In *IEEE Workshop on Directions in Automated CAD-Based Vision*, pages 115–123, Maui, Hawaii, June 1991.
- [31] D. Forsyth, J. L. Mundy, A. Zisserman, C. Coelho, A. Heller, and C. Rothwell. Invariant descriptors for 3-d object recognition and pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1991.
- [32] W. E. L. Grimson. The Combinatorics of Object Recognition in Cluttered Environments using Constrained Search. In *Proceedings of IEEE International Conference on Computer Vision*, pages 218–227, Tampa, Florida, December 1988.
- [33] W. E. L. Grimson. *Object Recognition by Computer-The Role of Geometric Constraints*. MIT Press, Cambridge, MA, 1990.
- [34] W. E. L. Grimson and T. Lozano-Pérez. Model-based recognition and localization from sparse range or tactile data. *International Journal of Robotics Research*, 3(3):3–35, 1984.
- [35] W. E. L. Grimson and T. Lozano-Pérez. Localizing overlapping parts by searching the interpretation tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(4):469–482, 1987.
- [36] L. Herault, R. Horaud, F. Veillon, and J.J. Niez. Symbolic Image Matching by Simulated Annealing. In *Proceedings of the British Machine Vision Conference*, pages 319–324, University of Oxford, England, September 1990.

- [37] R. Horaud, B. Conio, O. Le Boulleux, and B. Lacolle. An Analytic Solution for the Perspective 4-Point Problem. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, pages 500–507, San Diego, California, June 1989.
- [38] D. P. Huttenlocher and S. Ullman. Object Recognition using Alignment. In *Proceedings of the DARPA Image Understanding Workshop*, pages 370–380, Los Angeles, California, February 1987.
- [39] D. P. Huttenlocher and S. Ullman. Recognizing Solid Objects by Alignment. In *Proceedings of the DARPA Image Understanding Workshop*, April 1988.
- [40] D. P. Huttenlocher and P. C. Wayner. Finding Convex Edge Groupings in an Image. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, pages 406–412, Maui, Hawaii, June 1991.
- [41] K. Ikeuchi. Precompiling a geometrical model into an interpretation for object recognition in bin-picking tasks. In *Proceedings of the DARPA Image Understanding Workshop*, pages 321–339, Los Angeles, California, February 1987. Morgan Kaufmann Publishers, Inc.
- [42] A. Kalvin, E. Schonberg, J. T. Schwartz, and M. Sharir. Two-Dimensional, Model-Based, Boundary Matching Using Footprints. *International Journal of Robotics Research*, 5(4):38–55, 1986.
- [43] T. Kanade. Recovery of the three-dimensional shape of an object from a single view. *Artificial Intelligence*, 17:409–460, 1981.
- [44] E. Kishon and T. Hastie. 3-D Curve Matching Using Splines. In *Proceedings of European Conference on Computer Vision*, pages 589–591, Antibes, France, April 1990.
- [45] T. F. Knoll and R. C. Jain. Recognizing Partially Visible Objects Using Feature Indexed Hypotheses. *IEEE Journal of Robotics and Automation*, 2:3–13, March 1986.
- [46] J. J. Koenderink. *Solid Shape*. MIT Press, 1990.
- [47] D. J. Kriegman and J. Ponce. On recognizing and positioning curved 3-d objects from image contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1990.
- [48] Y. Lamdan, J. T. Schwartz, and H. J. Wolfson. On Recognition of 3-D Objects from 2-D Images. In *Proceedings of IEEE International Conference on Robotics and Automation*, April 1988.

- [49] Y. Lamdan and H. J. Wolfson. Geometric Hashing: A General and Efficient Model-Based Recognition Scheme. In *Proceedings of IEEE International Conference on Computer Vision*, pages 218–249, Tampa, Florida, December 1988.
- [50] T. S. Levitt and D. T. Lawton. Qualitative navigation for mobile robots. *Artificial Intelligence*, 1990.
- [51] R. Lewin. *Thread of Life*. Smithsonian Books, 1982.
- [52] D. G. Lowe. *Perceptual Organization and Visual Recognition*. Kluwer Academic Publishers, Hingham, MA, 1985.
- [53] D. G. Lowe. Three-dimensional object recognition from single two-dimensional images. *Artificial Intelligence*, 31:355–395, 1987.
- [54] D.G. Lowe and T.O Binford. Perceptual organization as a basis for visual recognition. In *Proceedings of AAAI-83*, Washington, D.C., August 1983.
- [55] D. Marr. *Vision*. W. H. Freeman and Company, 1982.
- [56] R. Mohan. *Perceptual Organization for Computer Vision*. PhD thesis, University of Southern California, August 1989. IRIS Technical Report 254.
- [57] R. Mohan and R. Nevatia. Using Perceptual Organization to Extract 3-D Structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(11):1121–1139, November 1989.
- [58] J. L. Mundy, A. J. Heller, and D. W. Thompson. The Concept of an Effective Viewpoint. In *Proceedings of the DARPA Image Understanding Workshop*, pages 651–659, Cambridge, Massachusetts, April 1988.
- [59] J. L. Mundy and D. W. Thompson. Model-Directed Object Recognition on the Connection Machine. In *Proceedings of the DARPA Image Understanding Workshop*, pages 93–106, Los Angeles, California, February 1987.
- [60] J. L. Mundy and D. W. Thompson. Three-Dimensional Model Matching from an Unconstrained Viewpoint. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 208–220, 1988.
- [61] H. Nasr, B. Bhanu, and S. Schaffer. Guiding an autonomous land vehicle using knowledge-based landmark recognition. In *Proceedings of the DARPA Image Understanding Workshop*, pages 432–439, Los Angeles, 1987.
- [62] R. Nevatia. *Machine Perception*. Prentice Hall, 1982.
- [63] R. Nevatia and T. O. Binford. Description and recognition of complex-curved objects. *Artificial Intelligence*, 8:77–98, 1977.

- [64] R. Nevatia and K. Price. Locating structures in aerial images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(5):476–484, September 1982.
- [65] H. Noltemeier. *Graphentheorie*. de Gruyter, 1976.
- [66] Horaud P. and Bolles R. C. 3DPO’s Strategy for Matching Three-Dimensional Objects in Range Data. In *Proceedings of the International Conference on Robotics*, pages 78–85, Atlanta, Georgia, March 1984.
- [67] B. Parvin and G. Medioni. A constraint satisfaction network for matching 3-D objects. In *Proceedings of the International Conference on Neural Networks*, volume II, pages 281–286, Washington, D.C, June 1989.
- [68] G. M. Radack and N. I. Badler. Local Matching of Surfaces Using a Boundary-Centered Radial Decomposition. *Computer Vision, Graphics, and Image Processing*, 45:380–396, 1989.
- [69] K. Rao and R. Nevatia. Computing volume descriptions from sparse 3-D data. *International Journal of Computer Vision*, 2:33–50, 1988.
- [70] L. G. Roberts. Machine Perception of Three Dimensional Solids. *Optical and Electro-Optical Information Processing*, pages 159–197, 1968.
- [71] H. Rom and G. Medioni. Hierarchical decomposition and axial shape description. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, 1992. (To appear).
- [72] P. Saint-Marc, J. S. Chen, and G. Medioni. Adaptive smoothing: A general tool for early vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):514–529, 1991.
- [73] P. Saint-Marc and G. Medioni. B-spline contour representation and symmetry detection. In *First European Conference on Computer Vision*, pages 604–606, Antibes, France, April 1990.
- [74] K. Sato and S. Inokuchi. Range-imaging system utilizing nematic liquid crystal mask. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 657–661, June 1987.
- [75] M. Seibert and A. M. Waxman. Adaptive 3-d object recognition from multiple views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1992.
- [76] A. Sha’ashua and S. Ullman. Structural saliency: the detection of globally salient structures using a locally connected network. In *Proceedings of IEEE International Conference on Computer Vision*, pages 321–327, 1988.

- [77] L. Stark and K. Bowyer. Achieving generalized object recognition through reasoning about association of function to structure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1991.
- [78] F. Stein and G. Medioni. Efficient Two Dimensional Object Recognition. In *Proceedings of International Conference on Pattern Recognition*, Atlantic City, New Jersey, June 1990.
- [79] F. Stein and G. Medioni. TOSS - A System for Efficient Three Dimensional Object Recognition. In *Proceedings of the DARPA Image Understanding Workshop*, Pittsburgh, Pennsylvania, September 1990.
- [80] F. Stein and G. Medioni. Map-Based Localization using the Panoramic Horizon. In *8th Israeli Symposium on Artificial Intelligence and Computer Vision*, pages 125–137, Tel Aviv, Israel, December 1991.
- [81] F. Stein and G. Medioni. Structural Hashing: Efficient Three Dimensional Object Recognition. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, pages 244–250, Maui, Hawaii, June 1991.
- [82] F. Stein and G. Medioni. Map-Based Localization using the Panoramic Horizon. In *Proceedings of IEEE International Conference on Robotics and Automation*, Nice, France, May 1992.
- [83] F. Stein and G. Medioni. Recognition of 3-D Objects from 2-D Groupings. *Proceedings of the DARPA Image Understanding Workshop*, January 1992.
- [84] F. Stein and G. Medioni. Structural Indexing: Efficient Three Dimensional Object Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):125–146, February 1992.
- [85] F. Stein and G. Medioni. Structural Indexing: Efficient Two Dimensional Object Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1992. (to appear).
- [86] R. Talluri and J. K. Aggarwal. Position estimation for a mobile robot in an unstructured environment. In *IEEE International Workshop on Intelligent Robots and Systems IROS '90*, 1990.
- [87] R. Talluri and J. K. Aggarwal. A positional estimation technique for an autonomous land vehicle in an unstructured environment. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space, i-SAIRAS '90*, 1990.
- [88] R. Talluri and J. K. Aggarwal. Position estimation for an autonomous mobile robot in an outdoor environment. *accepted for publication in IEEE Journal of Robotics and Automation*, 1992.

- [89] W. B. Thompson, H. L. Pick, B. H. Bennett, M. R. Heinrichs, S. L. Savitt, and K. Smith. Map-based localization: The drop-off problem. In *Proceedings of the DARPA Image Understanding Workshop*, 1990.
- [90] C. Thorpe, S. Shafer, T. Kanade, and M. H. Hebert. Vision and navigation for the carnegie mellon navlab. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1988.
- [91] R. Y. Tsai. A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology using Off-the-Shelf TV Cameras and Lenses. *IEEE Journal of Robotics and Automation*, 3(4):323–344, 1987.
- [92] L. W. Tucker, C. R. Feynman, and Fritzsche D. M. Object Recognition Using the Connection Machine. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, pages 871–878, Ann Arbor, Michigan, June 1988.
- [93] J. R. Ullman. An algorithm for subgraph isomorphism. *Journal of the Association for Computing Machinery*, 23(1):31–42, 1976.
- [94] S. Ullman and R. Basri. Recognition by linear combinations of models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1991.
- [95] F. Uluþınar and R. Nevatia. Recovering shape from contour for constant cross section generalized cylinders. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, pages 674–676, Maui, Hawaii, June 1991.
- [96] Jeffrey S. Vitter and Wen-Chin Chen. *Design and Analysis of Coalesced Hashing*. Oxford University Press, 1987.
- [97] A. Witkin and J. Tenenbaum. On the role of structure in vision. In J. Beck, B. Hope, and A. Rosenfeld, editors, *Human and Machine Vision*, pages 481–543. Academic Press, New York, 1983.
- [98] J. Y. Zheng, M. Barth, and S. Tsuji. Qualitative route scene description using autonomous landmark detection. In *Proceedings of IEEE International Conference on Computer Vision*, December 1990.