INDEXING WITHOUT INVARIANTS
IN
MODEL-BASED OBJECT RECOGNITION

by

Jeffrey S. Beis

M.Sc., The University of Toronto, 1988

B.Sc., Dalhousie University, 1985

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF COMPUTER SCIENCE

We accept this thesis as conforming
to the required standard

The University of British Columbia

July 1997

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of COMPUTER SCIENCE

The University of British Columbia
Vancouver, Canada

Date JULY 30, 1997

DE-6 (2/88)

Abstract

This thesis presents a method to efficiently recognize 3D objects from single, 2D images by the use of a novel, probabilistic indexing technique. Indexing is a two-stage process that includes an offline training stage and a runtime lookup stage. During training, feature vectors representing object appearance are acquired from several points of view about each object and stored in the index. At runtime, for each image feature vector detected, a small set of the closest model vectors is recovered from the index and used to form match hypotheses. This set of nearest neighbours provides interpolation between the nearby training views of the objects, and is used to compute probability estimates that proposed matches are correct. The overall recognition process becomes extremely efficient when hypotheses are verified in order of their probabilities.

Contributions of this thesis include the use of an indexing data structure (the kd-tree) and search algorithm (*Best-Bin First* search) which, unlike the standard hash table methods, remain efficient to higher index space dimensionalities. This behavior is critical to provide discrimination between models in large databases. In addition, the repertoire of 3D objects that can be recognized has been significantly expanded from that in most previous indexing work, by explicitly avoiding the requirement for special-case invariant features. Finally, an incremental learning procedure has been introduced which extracts model grouping information from real images as the system performs recognition, and adds it into the index to improve indexing accuracy. A new clustering algorithm (*Weighted Vector Quantization*) is used to limit the memory requirements of this continual learning process.

The indexing algorithm has been embedded within a fully functional automatic recognition system that typically requires only a few seconds to recognize objects in standard sized images. Experiments with real and synthetic images are presented, using indexing features derived from groupings of line segments. Indexing accuracy is shown to be high, as indicated by the rankings assigned to correct hypotheses. Experiments with the Best-Bin First search algorithm show that, if it is acceptable to miss a small fraction of the exact closest neighbours, the regime in which kd-tree search remains efficient can be extended, roughly from 5-dimensional to 20-dimensional spaces, and that this efficiency holds for very large numbers of stored points. Finally, experiments with the Weighted Vector Quantization algorithm show that it is possible to incorporate real image data into the index via incremental learning so that indexing performance is improved without increasing the memory requirements of the system.

Table of Contents

List of Tables

List of Figures

Acknowledgements

This dissertation owes much to the optimism, perspicacity, responsibility, and various other kinds of supportive adjectives I could apply to David Lowe, who has been my supervisor. He is an outstanding researcher, and the little I do know regarding the strange process of doing research has been learned in large part from him.

I would like to thank the other members of my supervisory committee: Jim Little, Raymond Ng, and Bob Woodham. Their comments were instructive and helpful, and I greatly appreciate the time they committed to do the job that they did. Thanks go as well to my University Examiners, David Poole and Jim Enns, and my External Examiner Gerard Medioni, for their efforts and insights.

Throughout the many years (decades?) of my tenure, discussions with several graduate students kept things interesting. Mike Sahota was always asking particularly annoying (i.e. difficult) questions, and John Lloyd, Jiping Lu, Dan McReynolds, Richard Pollock, Chris Romanzin, and Vlad Tucakov were all helpful and provided valuable feedback at different times. They also contributed to the overall intellectual level of the department, both within the realm of research and beyond, and I figure this was important for my overall sanity.

The support staff both within the LCI — Val, Dan, and even Rod and Stewart if the wind was blowing in the right direction — and the department made life and bureaucracy a lot easier to get through.

The final year and a half would have been absolutely miserable had it not been for close friends Lynn Booth, John Crusius, Joel Chauvin, Mike Magee, Jill Thomas, and The Gang: Cristina, Tarik, Ivailo, Katja, Cristof, and Ken.

Financial support was provided by the National Sciences and Engineering Research Council of Canada, as well as by the Institute for Robotics and Intelligent Systems, one of the Canadian Networks of Centres of Excellence.

I dedicate this dissertation to my parents

Frances and Richard

Chapter 1

Introduction

To do object recognition by computer is to automatically determine which regions of an image correspond to known entities in the real world. The fundamental step in this process is matching, that is, matching some portions of a known object model with some segments of the image, so that the set of matches is consistent with the object's structure. The level at which the search for correspondences takes place varies from approach to approach — features may be more or less complex, for example points or edges, image curves, full silhouettes, or patches of raw intensity data. Given a large enough set of matches, we may become convinced that an instance of an object has been detected. The complexity of the object recognition problem in computer vision lies in the astronomical number of possible combinations of model-to-image feature matches that must be considered.

Indexing is a way to dramatically speed up the search for matches. In simple terms, an index is a mechanism which, when provided with a key value, is able to rapidly access some associated data. In object recognition, image shapes are the keys, and the indexing process recovers model shapes that could have generated them. Instead of having to consider all possible matches and explicitly reject invalid ones, indexing inverts the process so only matches that are known to be plausible are considered. This is accomplished by a reordering of the data, from its original organization within a 3D model database, to the index ordering which is according to

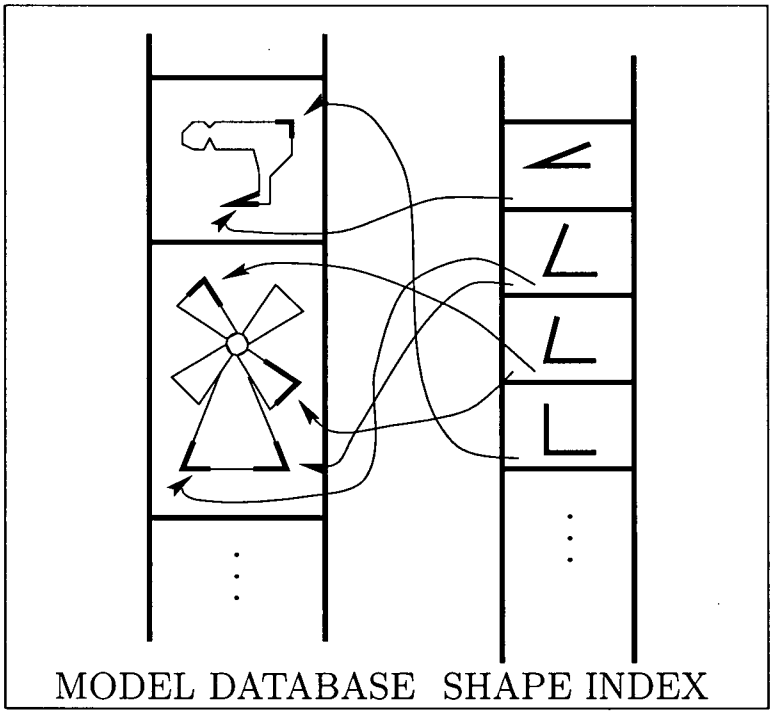


Figure 1.1: Indexing as Remapping (schematic). Left array shows model database ordered by object label. Right array shows index ordered by feature value, which simplifies the search for shape matches. In reality, objects are 3-dimensional, and index shapes are much more complex.

a numerical descriptor for 2D image shapes (see Figure 1.1). By making shapes more complex than those shown in the figure, only a small number of model correspondences will be indicated by each index entry.

This problem is difficult for several reasons. The foremost of these is that the image projection of a 3D shape varies continuously with viewpoint. This implies that each underlying 3D shape in general cannot be represented as a single entry within the index. In fact, the projection process not only means that each 3D shape corresponds to many 2D shapes, but that any 2D image shape may be due to a variety of 3D shapes. In other words, any local image shape can never be attributed to a single model with absolute certainty. This ambiguity of image shape is exacerbated by the fact that different objects may even share identical subparts. The goal of indexing, therefore, is not only to rapidly uncover potential matches, but also to

assign a probability to each interpretation. A further problem is that because of the complexity of the imaging process, it is difficult to accurately model. Generating an exact set of object appearances to store in the index using only synthetic object models is therefore troublesome.

This thesis provides a solution to the indexing problem that explicitly deals with the problems of shape variation, ambiguity, and actual versus modelled appearance. The methodology addresses the primary concerns of an index, which are accuracy and speed. Accuracy is the ability of the index to reduce the set of possible matches to a very small number, without eliminating the valid matches. Speed refers to the index lookup time, which must be kept small so as not to offset the reduction in verification time due to the index accuracy.

Our solution consists of sampling the view sphere for each 3D model shape and storing the generated set of 2D shape projections in the index. Each sample is represented as a real-valued, multi-dimensional feature vector (equivalently, a point in a multi-dimensional feature space). The data are stored in a kd-tree, and we present an approximate nearest neighbour search algorithm that efficiently recovers the closest points to any runtime shape query. This algorithm remains efficient in high-dimensional spaces and with very large numbers of points.

The recovered set of neighbours might, for example, contain a few samples from each of a few different models. The number of samples that indicate a given model, and how similar each sample is to the query, can be used to estimate the probability that the model could have given rise to the image shape. Thus, the index is effectively storing probability distributions of possible correspondences, rather than simple correspondence hypotheses. A ranking stage is appended to the indexing stage to order the hypotheses according to their probability estimates, so that the most likely can be investigated first. Finally, as the indexing stage is necessarily probabilistic, a verification stage is required, and it uses considerably more information to ensure that the final recognition decision is correct.

Storing distributions as sample sets rather than in analytical form is necessary, since the actual distributions are not available, but it also makes index updates straightforward. Simply adding a new point belonging to a particular model changes its *a priori* probability with

respect to other models, and at the same time re-shapes the model's internal distribution. An incremental learning stage is proposed in which image data is added to the index when objects are recognized, so that eventually the index representation of an object will be dominated by the features detectable in real images. For such a continuous learning system to be practical, a clustering algorithm is required to keep the number of samples per grouping small. We suggest a simple approach that reduces the number of stored points while attempting to minimize the distortion to the probability estimates.

The indexing mechanism reduces the number of model matches that must be considered for a given image shape. By a suitable choice of features, it is also possible to reduce the absolute number of image shapes to consider. In this thesis the primitive feature is the straight edge segment, and the shapes stored in the index are particular groupings of edges, which exhibit specific properties that are unlikely to occur in an image by chance. Figure 1.2 presents an example of the type of recognition problem considered in this thesis, and its solution (originally outlined in (Beis & Lowe 1994)). Two of the types of grouping used for indexing can be seen in the third panel.

Certain aspects of the method should be noted.

1. **Non-invariance:** Most indexing methods assume that invariant shape descriptors exist, and this significantly simplifies the problem. However, there are no general invariants under 3D perspective projections, so we have chosen to store probability distributions for 2D shapes instead. If invariants are available for certain objects, they can easily be accommodated within our approach. In those cases, the probability machinery might be useful to learn the inevitable spread of the "invariant" due to random noise, and possibly to structural noise that may be correlated with viewpoint.

2. **Probabilistic:** This approach is by nature probabilistic. Grouping is probabilistic in that, by restricting feature sets to certain non-accidental configurations, it is always possible to

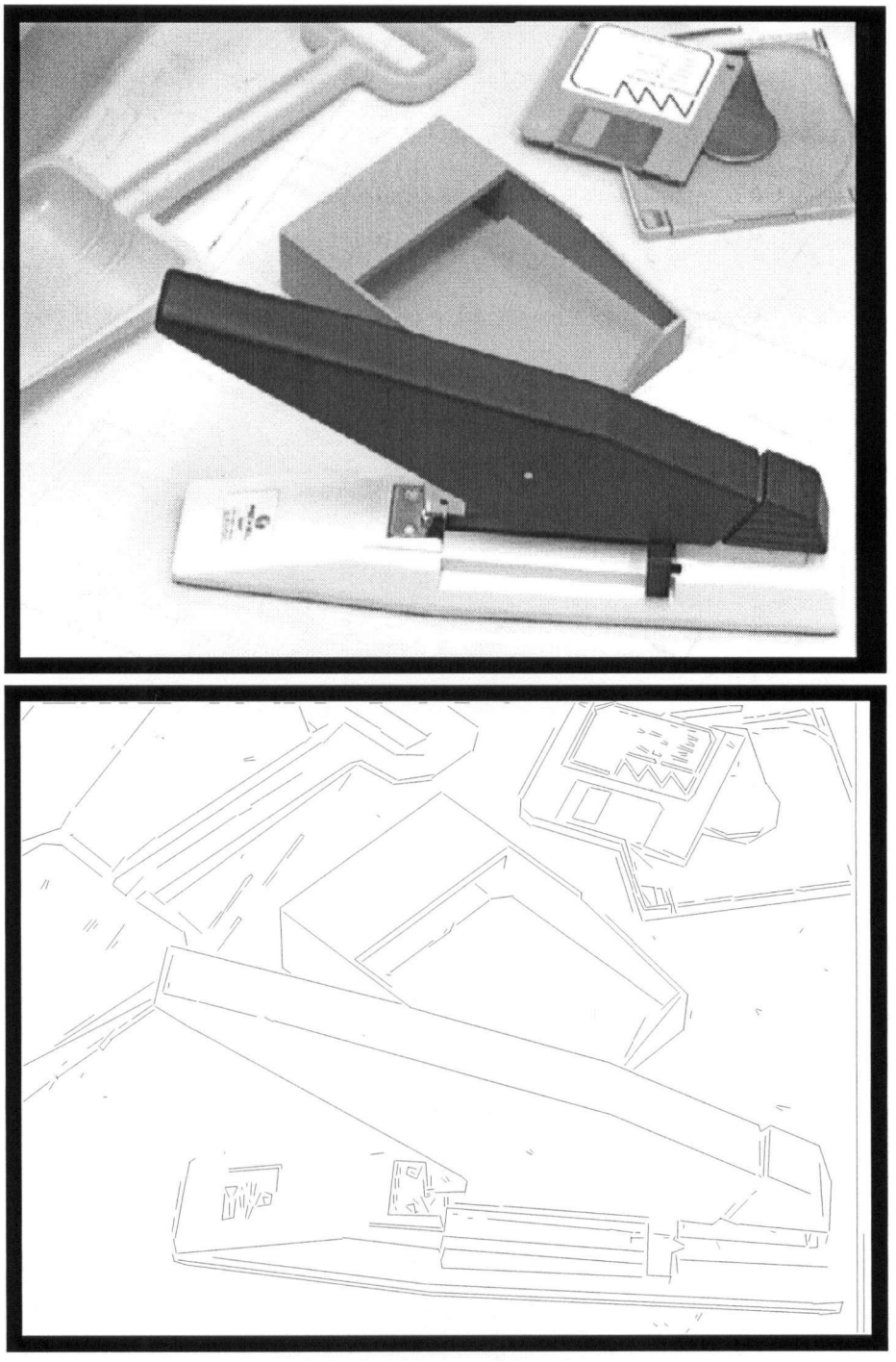


Figure 1.2: Upper panel: Original grey-scale image. Lower panel: Edge-detected image used for indexing and matching.

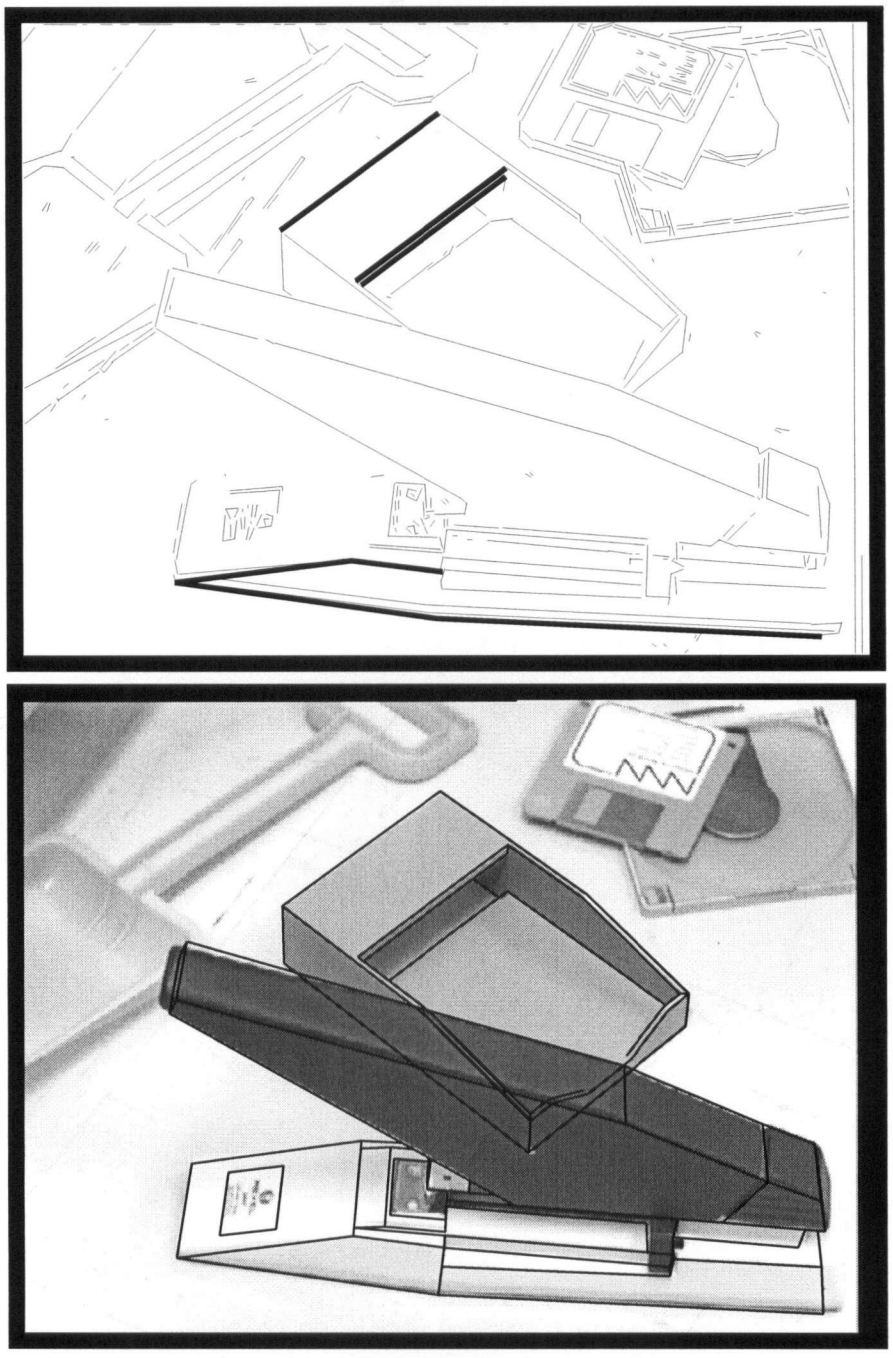


Figure 1.2: (cont'd). Upper panel: The initial shape matches suggested by the index are shown in bold, including a 3-segment parallel grouping for the noteholder and a 4-segment chain grouping for the stapler. The number of hypotheses investigated before both objects were verified was < 100. Lower panel: Correctly recognized objects overlaid on the original image.

miss other valid combinations. Efficient nearest neighbour recovery from the kd-tree will also be shown to be probabilistic, occasionally failing to discover the closest neighbour(s). These trade-offs are crucial for efficiency, but there is no guarantee that a more thorough search would not have succeeded. To make the possibility of a miss very small, redundancy of groupings is used — there should be a rich and extensive set of groupings for any view of an object to ensure successful indexing.

3. **Generality:** In this thesis a specific shape primitive, the straight edge, was chosen for demonstration purposes, and all groupings used in the experiments were combinations of line segments. The method is much more general than this choice might imply, and for a feature to be useful the sole requirement is that it can be quantified as a real-valued number. The potential therefore exists to use curved edges, texture patches, and color regions as features. Some ideas for integrating color with the current set of geometric features are presented in the section on future work.

4. **Image noise:** Instead of explicitly modelling the imaging process, all aspects of noise are to be handled by the sampling of shape distributions in real images. Although our algorithm is currently bootstrapped using synthetic model training data, we later introduce an update stage that incorporates real image data into the index. Thus, instead of assuming a necessarily simple noise model that applies to all feature sets, the system can learn the noise characteristics of each one individually.

5. **Pose:** In contrast to methods which represent objects as sets of 2D images, in our approach an accurate pose estimate is available once an object has been recognized.

1.1 Thesis Organization

Chapter two goes into detail regarding the problems of indexing a database of 3D models using 2D image data. A review of previous indexing research is given.

Chapter three presents the solution to the indexing problem adopted by this thesis.

Chapter four gives details of the implementation of the indexing system and of the object recognition framework in which it is embedded. Experiments are reported which demonstrate the speed and accuracy of the proposed algorithms using synthetic and real images of varying complexity.

Chapter five discusses a new approximate algorithm for nearest neighbour recovery in high-dimensional spaces called "Best Bin First" (BBF) search. Experiments are presented which imply that our indexing method scales well both with the dimensionality of the index space and the number of points (or models) stored.

Chapter six looks at the practical issue of limiting memory growth as new data is incorporated into the index. A new clustering algorithm called "Weighted Vector Quantization" (WVQ) is formulated which is designed to reduce memory requirements while minimally distorting probability estimates. Experiments show that WVQ compares well to previous methods.

Chapter seven outlines an approach to semi-automatic construction of object models, and provides an example of a model built from example images as a proof-of-concept.

Chapter eight presents the most important avenues for future work on this topic.

Chapter 2

The Problem of Indexing

2.1 An Index for Shapes

At the most abstract level, the defining properties of an index are that it is

(a) a list of items, or keys, that are

(b) stored in some useful ordering, and

(c) the list entries point to the original ("unordered") locations of the items.

In a book, for example, the index items are topics or key words, the order is alphabetical, and the pointers are to the pages where the items appear. Given a key word, it is much faster to use the index than to scan the entire text for instances of the word. In this example, indexing provides a reduction from a linear to a logarithmic time search (assuming a binary search to locate the key word in the index). In object recognition, the primitives have to do with shape and appearance. Index keys are features or feature groupings, the ordering is according to numerical values assigned to the features, and the entries point to the spatial locations of the features on the models in the database.

To understand the context in which indexing is generally applied, we must consider the historical approaches to model-based object recognition, where the main paradigm has been

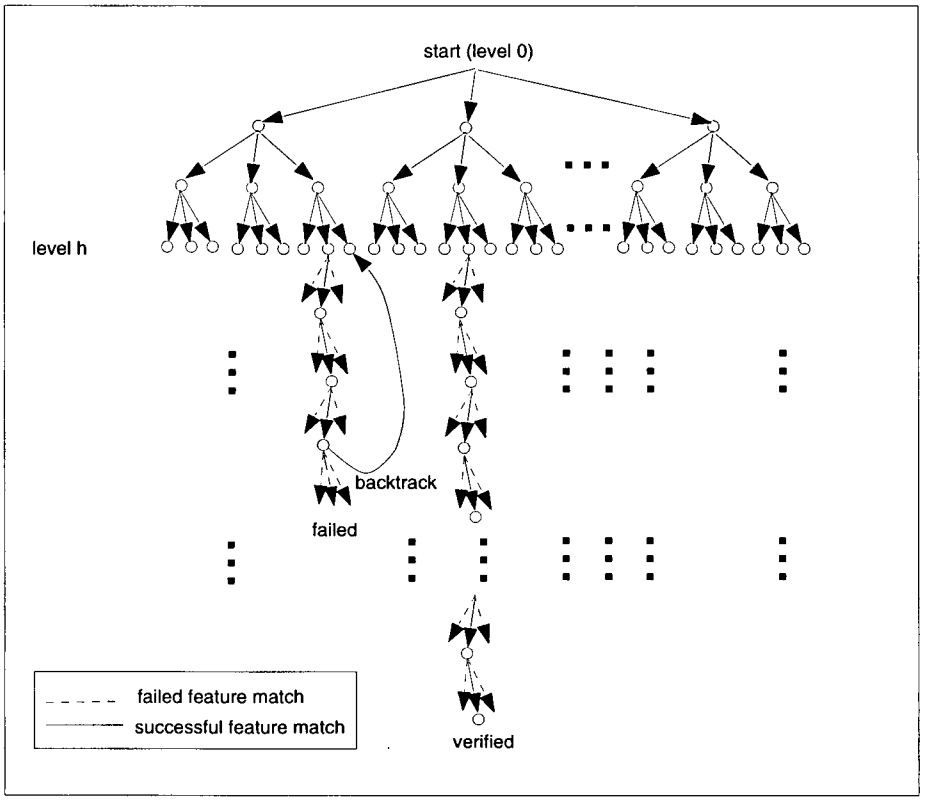


Figure 2.1: Hypothesize-and-test tree search (similar to figure 9, p.1544 of (Chen & Kak 1989)). Each path from root to level "h" contains enough feature matches to completely constrain the model pose. Back-projection of the model into the image leads to strong constraints on further matches, and allows immediate pruning of most branches.

constrained tree search. For each model, an "interpretation tree" of possible feature matches consists of each model feature matched with each image feature. A path from the root to a leaf node corresponds to a complete interpretation of a model. If N is the number of image features and M the number of model features, the worst-case time complexity for these depth-first searches is $\mathcal{O}(N^M)$.

Simple pairwise constraints between features are effective in pruning the tree (Grimson & Lozano-Perez 1987, Ayache & Faugeras 1986), but for complex models and images the exponential complexity makes the algorithm impractical. The *alignment* algorithm (Huttenlocher & Ullman 1987) improves on this with a clever way of pruning. The tree is expanded only to a

depth such that each path contains just enough matches to fully constrain the model pose (level "h" in Figure 2.1). A verification stage is applied in which the model is back-projected into the image using the computed pose and the image is searched for additional matches. The strong constraints provided by the image data allow most of the hypotheses to be quickly eliminated, with only a small fraction requiring further expansion of the tree. The complexity is thus reduced to $\mathcal{O}(N^3M^3)$ per model, assuming orthographic projection and point features, which is still impractical for most recognition tasks. The general approach of hypothesizing a match and applying a verification stage to remove false positives is known as "hypothesize-and-test".

In this context, indexing avoids the need to fully expand the tree even to the alignment level. In a preprocessing stage the appearance of feature groupings is predicted and stored in the index. At runtime, for any image feature set the index points directly to those nodes in the tree that would not be pruned, i.e., that are known to provide a feasible match down to level $h+1$ or greater, depending on the complexity of the key. Groupings must point at least one level deeper than the minimal alignment depth, otherwise any two groupings can be matched and there is no reduction from exhaustive comparison. For greater specificity, indexes should point deeper into the tree.

This process leads to speed improvements in two ways. Firstly, for a single model, indexing accomplishes the type of massive, efficient pruning of hypothetical matches just mentioned. Secondly, tree search methods must consecutively search one tree for each model. With the use of an index, each entry can have pointers to all occurrences of a shape on all models, so a sequential search through the set of models is avoided.

The cost involved in indexing is twofold. The first is the preprocessing stage required to build the index. Items corresponding to the appearance of all models from all points of view must be generated and stored in the index, and this process can take a significant amount of computing time and memory. However, the time taken to build the index is not considered critical, since it is done only once, it is done offline, and does not affect runtime performance.

The more important cost is the actual time taken to recover hypotheses from the index at runtime. The index data structure may become quite large, as it is desirable that none of the shape information from the original models is omitted, so the lookup time will be a real consideration. However, this cost is more than offset by the reduction in verification time from not having to explicitly reject the enormous number of hypotheses that indexing effectively prunes.

The process of indexing can be seen as a re-mapping, from a search space with no useful ordering into one that is structured to solve a particular search problem. There is a significant body of work available that investigates the problem of searching such ordered data, and we will see that it can be used to build an effective indexing system, which solves the recognition problem much more efficiently than it can be solved in its original form. In Chapter 4 we show that indexing reduces the complexity of recognition to $< \mathcal{O}(NM)$.

2.2 The Problems of Shape-based Indexing

While indexing for object recognition shares the same core components as any other type of indexing, there are several issues that are particular to this application. Before listing these traits, note that in what follows the term "shape" will be used instead of "appearance" for convenience. Because the use of geometric features is much more developed in the object recognition community than, say, color or texture, the association of "features" with "shape" will be more natural for most readers.

Problem 1: *Shape is continuous.* Indexing is typically applied to discrete data, where keys are integers or text strings. The natural keys for our indexing task are shapes, and shapes are more naturally represented as real-valued numbers than as integers. An example of such a feature would be the angle, which can take on any value in the continuous range $[0, 2\pi]$. An indexing system must therefore either discretize feature values, if standard procedures are to be applied directly, or else determine a new way to handle continuous data.

Figure 2.2: "Shape is ambiguous". A single image grouping (white lines) indicates several possible model matches (black lines) via indexing. At most one of these may be valid. Can indexing be used to indicate which of these hypotheses is most likely?

Problem 2: *Shape is ambiguous.* A single set of image features may suggest a match with more than one object (Figure 2.2). This is clear since parts of different objects may in fact have identical image appearances. The problem is especially relevant for the 3D-from-2D recognition problem, as a wide variety of 3D shapes can result in the same 2D shape after projection, depending on the viewpoint of the camera. Suppose that a particular image shape potentially matches several of the database objects. We would like to know if there is a way for an indexing process to indicate, prior to a verification procedure, which of the recovered hypotheses is most likely to be correct?

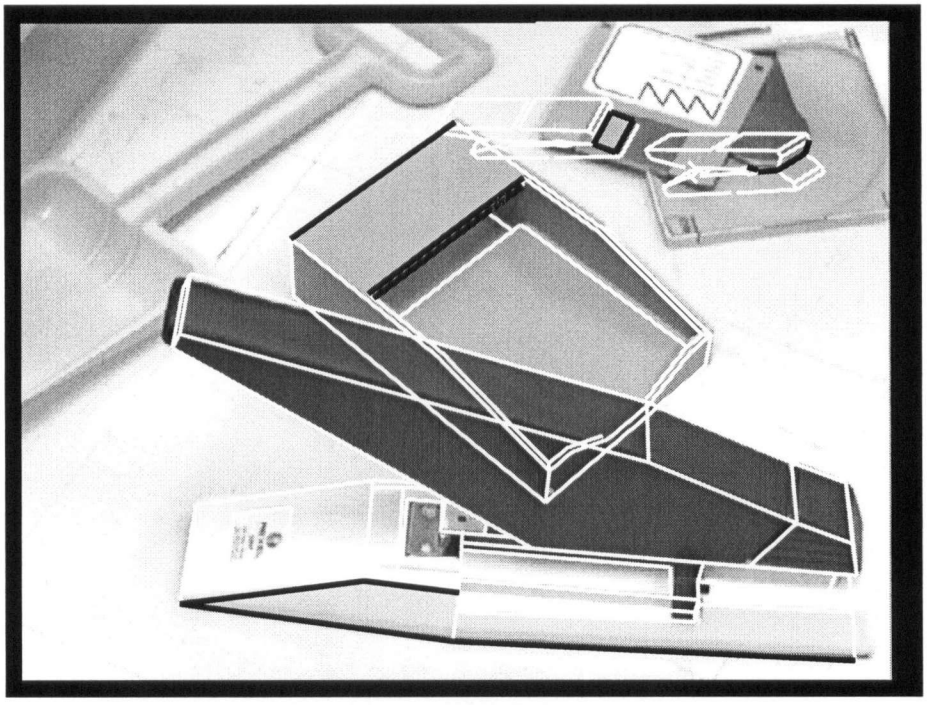


Figure 2.3: "Data is ambiguous". Several feature groupings detected in the image (black lines) indicate possible model matches (white lines) via indexing. Only very few of these may be valid. Can indexing be used to indicate which of these hypotheses is most likely?

Problem 3: *Data is ambiguous.* A single image will contain multiple feature sets, so the index must be accessed multiple times. In general, some of these queries will lead to correct hypotheses, and others will not (Figure 2.3). Supposing that one of several indexed matches

from a single image is valid, we would like to know if there is a way for an indexing process to indicate, prior to a verification procedure, which of the hypotheses is most likely to be correct? To see that this is different from Problem 2, suppose that shape matches were not ambiguous with respect to the model database (any image shape could only imply a match with one model feature set), yet still probabilistic (the implied match might or might not be valid). Then it would still be important to know which of the many image feature groupings would provide the most likely match.

Problem 4: *Shape varies with viewpoint.* This phenomenon is due to the projection from 3D to 2D. Even if the same object faces are visible from different viewpoints, some of the angles on the faces will have changed in the image. The problem is non-trivial since the variation is both highly non-linear and continuous, the latter implying that there is an ostensibly infinite set of appearances for each object in the database. In other forms of the recognition problem, the 2D-from-2D and 3D-from-3D versions, this difficulty does not exist.

Problem 5: *Shape is not already quantified as a number.* What is the best shape descriptor to use for indexing? At a minimum it must have the property that similar shapes lead to similar index values, since noise will cause small variations in the feature values. Ideally, the shape descriptor would be invariant to the change in viewpoint, so only a single index entry would be needed for each 3D model shape.

Problem 6: *Data is noisy.* Despite an object being in full view of a camera, noise is one further reason why a shape-based indexing technique may have difficulty locating it.

(a) *Missing data.* Occlusion may conceal some portion of a known object. Furthermore, depending on the viewpoint and lighting conditions, a 3D feature that typically produces an image feature may fail to do so. One example would be an edge that becomes invisible when there is a lack of contrast from one side to the other. (This problem could be relabelled "(perceived) shape varies with lighting conditions".)

(b) *Inaccurate data.* Feature values may be slightly distorted from one image to another depending on lighting conditions, digitization, and edge detector properties. In the case of edges, feature detectors may break a single edge into several smaller edges, or confound edges from two (or more) objects and so generate a single edge.

In addition to the above problems, the fundamental properties that determine the performance of an index are the same as for any other computer algorithm: time and space efficiency. Time efficiency is the main reason to use indexing in the first place. This can be broken into two regimes, the time required to build the index and the time required to access the index. The former is relatively unimportant up to a point: it is convenient if the index can be recomputed quickly, but for an algorithm with good runtime performance, a very long training stage would be acceptable. The ability to rapidly update the index is particularly useful when new models, or new data for the current models, must be added on a regular basis. For most applications, however, it is the *access time* that will be most critical, since ideally the recognition algorithm should run in real time. It is difficult to translate this into a statement about bounds on acceptable algorithmic complexity, because in computer vision it is the *absolute* time requirements which are more important than the theoretical limiting cases.

The trade-offs made by indexing to achieve runtime speedup include not only the time spent building the index, but also the space needed to store it. Because an index must represent each object as it appears from any point of view, the amount of data to store can become very large. This is one reason that invariant shape descriptors are desirable. However, because memory is rapidly becoming cheap and immense, space may be more of an indirect than a direct issue, in that access time will be tied to the size of the index.

2.3 Previous Indexing Research

2.3.1 Invariant Indexing

This class of methods is designed specifically to solve **Problem 4** above, the variation of shape with viewpoint. The idea is that, while the projected image shape indeed varies with viewpoint, there may be *computable functions* of the shape which do not. The important question when searching for invariant features is "what is the set of possible transformations a feature may undergo?" For example, a 2D world involves only translations and rotations in the plane. Thus choosing point locations for features is bad (non-invariant) while angles (invariant) are good. Formally, if the set of possible transforms is $\mathcal{T}$ and $\vec{\mathbf{x}}$ is a feature vector describing the underlying shape, then an invariant measure $\mathcal{I}$ has the property that $\mathcal{I}(\mathcal{T}_1(\vec{\mathbf{x}})) = \mathcal{I}(\mathcal{T}_2(\vec{\mathbf{x}})) \; \forall \; \mathcal{T}_1, \mathcal{T}_2 \in \mathcal{T}$.

If invariants do exist for particular models, there are huge benefits. A single index entry can represent all possible appearances of the underlying shape. This not only keeps memory costs at a minimum, but simplifies the training stage dramatically, because only one view of each invariant feature is required for a complete model representation within the index.

Often, however, unrealistic demands for particular types of feature configuration severely limit which objects can be recognized. For example, a typical requirement would be that there are several coplanar features available, which is perhaps only true for polyhedral objects with surface markings. Invariant indexing methods can be further classified according to whether models are 2D or 3D, whether image data is 2D or 3D, and which model of projection is assumed: orthographic, weak perspective, or full perspective.

2D-from-2D

The case of indexing 2D models using 2D data is relatively straightforward because simple properties like angles between segments, and relative edge lengths of segments, are invariant to 2D rotations, translations, and scaling.

Wallace (1987) uses a set of four simple, binary relationships between pairs of features to generate 2– and 3–dimensional index spaces. For example, two straight edges define a 3D index space representing the angle between the segments and the distances from the midpoint of each segment to the intersection point of virtual lines lying on top of the segments. Hypotheses recovered from the index, which indicate a single object, are collected and used to structure a tree search, rather than directly in a hypothesize-and-test approach.

Stein & Medioni (1992a) use larger feature groupings, which they call "super segments", to compute their index. These are chains of coterminating straight line segments generated by polygonal approximations to image curves. Multiple scales of smoothing are applied to the digital curves, creating a series of approximations. The complete set is used during storage and retrieval, in an attempt to mitigate the effects of the distortion caused by approximating curves with straight lines. The multi-dimensional index vectors in this method are comprised of the angles between consecutive segments plus the "eccentricity" of the entire group, which is the second moment of inertia of the vertices of the super segment. Our method also uses segment chains as one type of grouping to generate index vectors. Instead of eccentricity, however, we use length ratios of the adjacent chain segments. These features contain more information than eccentricity, and unlike eccentricity uniquely constrain (in conjunction with the angle features) the shapes which correspond to them.

3D-from-3D

In this class, image information is in the form of a depth map, from an imaging source such as a laser range finder, stereo camera system, or tomographic reconstruction. Choosing invariants is straightforward for the same reason as in 2D-from-2D indexing: absolute lengths and angles between 3D edges and planes are invariant to 3D rotations and translations; angles and relative lengths are invariant to 3D rotations, 3D translations, and to scalings.

Flynn & Jain (1992) provide what is essentially the 3D equivalent of Wallace's system. Simple, invariant, pairwise relations (angles, distances) between triples of primitives (planes,

cylinders, spheres) are used to generate ten 2D index tables. An example of a feature group is a set of 3 planar faces. The 2D index vector corresponding to this set consists of two angles, between one plane's normal vector and the normals of the other two planes.

Stein & Medioni (1992b) adapt their 2D work for use with 3D range data. They use two types of feature: super-segments, generalized to 3D, and, for surfaces that do not have good polyhedral segmentations, a novel shape descriptor for local surface patches, which they call a "splash". Splashes describe the local surface orientation about a given surface point. Interestingly, splashes can be encoded in exactly the same form as a super segment, so that the same data structure can be used to store the index for both types of feature.

3D-from-2D

Difficulties arise when we address the case where models can be fully 3-dimensional, but the image data is only 2D. This means adding projection to the set of transformations over which we desire invariant measures. Recall that the invariances in the previous sections pertained only to rotations, translations, and scalings. While there are no invariants for *general* 3D feature sets projected into 2D (e.g. (Clemens & Jacobs 1991)), there has been a fruitful mining in the area of *constrained* 3D feature sets. These are special configurations of shapes/features from which invariant features may be computed.

A common restriction has been to demand that all features lie in a plane. Forsyth, Mundy, Zisserman & Brown (1990) describe a set of invariant measures which use this constraint and assume a weak perspective approximation to full projection. These are computed from, among other things, pairs of planar conics, and sets of 4 coplanar lines which meet at a point (the familiar cross-ratio invariant). Rothwell, Zisserman, Mundy & Forsyth (1992) derive their invariants from concavities in planar curves. Four special points can be defined for any concavity, and these are used to set up a canonical reference frame. An image curve is transformed to this frame, where it takes on a unique, invariant form. In the canonical coordinate system, the curve together with the x-axis form a closed region, and area moments are used as the invariant

measures. This system can handle full perspective projection. Wayner (1991) also uses the co-planarity constraint for one invariant, requiring two triangles to be coplanar on the 3D model. A second invariant can be calculated under the assumption that four 2D scene points represent three orthogonal vectors in 3-space (which is, however, impossible to determine from the image data).

These types of invariants are useful if the objects in the database contain the necessary features. However, the restrictions mean that the set of objects that can be indexed is quite limited. Typical examples of objects that can be recognized by these methods include flat tools (scissors, wrench), puzzle pieces, or oddly-shaped flat machined parts used in manufacturing. For this reason, and because our domain of interest is 3D-from-2D indexing, in our approach we will choose not to rely on invariants for indexing. Of course, if there are known invariances, there is nothing in our approach which prevents us from exploiting them in order to reduce storage.

3D-From-Multiple-2D

These are methods which use two or more 2D images to recognize 3D objects. Barrett, Payton, Haag & Brill (1991) provide a mathematical treatment appropriate for projective transforms. As a simple example, they derive the familiar cross-ratio invariant for planar feature sets. Their "ratio-of-determinants" approach to finding invariants can be applied to unrestricted 3D point sets, but may require correspondence of up to 8 point features in *34 images.*

Another method requiring multiple images is that of Weinshall's group (Weinshall 1993, Weinshall & Tomasi 1993, Mohan, Weinshall & Sarukkai 1993, Shashua 1993). The invariants are matrices, the elements of which are functions of an ordered set of point coordinates. So, for example, Mohan et al. (1993) need 4 points tracked across 3 images to derive a projective invariant that is a matrix with 6 independent elements. This type of invariant (i.e., a matrix) can be used for indexing simply by using a canonical ordering of the N matrix elements to generate a point in an N-dimensional index space.

Differential Invariants

All invariants described above fall within the class of the algebraic invariants. Differential and semi-differential invariants may also be useful for indexing. Work in these areas has not progressed to the stage of implementation in real vision systems, but the following short review is included for the sake of completeness.

Weiss (1988, 1992) pioneered the use of differential invariants in the arena of object recognition. These invariants can be calculated at any point along a planar curve, and require high (up to 4th) order derivatives of the curve at that point. The invariants are coefficients of polynomials fit to the region of the specified curve point. The method is mathematically satisfying, but practically questionable due to the instability in the derivatives of digitized curves. Furthermore, even if we can acquire very smooth, high-resolution curves, the sensitivity of the polynomial coefficients to small changes in shape has not been investigated.

Semi-Differential Invariants

Semi-differential methods trade off point correspondences for orders of derivative required to compute an invariant. An oft-cited paper by Van Gool, *et al.* (1991), uses Lie group theory to derive an equation governing the invariants of any function $f : R^m \rightarrow R$, where the components of R^m are taken to be both coordinates of image-curve points and derivatives of the curves at these points. The invariants suggested are derived from closed boundary curves, which are not practical in the case of occlusion. Bruckstein, Holt, Netravali & Richardson (1993) use a similar mathematical approach. They suggest some more local, semi-differential measures that could be used for indexing, but no tests are done with images of any type.

2.3.2 Non-Invariant Indexing

This is the most general form of indexing, and most closely related to the proposed approach. No assumptions are made that invariant features exist for each model. Instead, the index is

built using many views of an object, so that it can be recognized when seen from any direction. More memory is needed for these methods than for invariant indexing methods, but more types of object can be recognized, i.e., those that do not contain invariant feature sets.

A strong justification for this approach is that there are no invariants for arbitrary 3D point sets projected into two dimensions (Clemens & Jacobs 1991, Barrett et al. 1991, Burns, Weiss & Riseman 1993). This is true even for the simplest model of projection (orthographic). Instead, the view variation of these feature sets produces 2D sheets in index spaces of dimension 4 and higher (whereas an invariant would give a single point). A clever manipulation by Jacobs (1991) allows each 2D sheet to be decomposed into two 1D lines in a pair of related 2D index spaces. This sets a lower bound on the amount of space required by any general 3D-from-2D indexing method that uses hash table lookup. Unfortunately, even in this case memory usage can be extensive: for a discrete lookup table (LUT), there must be an entry in each of the bins through which a sheet (or line) passes. Furthermore, the desired sampling density in the index space (i.e., the bin size) is not readily translated into a sampling density on the view sphere[1] because the mapping from viewpoint to feature appearance is highly non-linear.

An early influential paper in the area of non-invariant indexing is (Thompson & Mundy 1987). They create LUTs which can be used to access the viewing transform between a detected "vertex-pair" feature in an image and a matching pair from a 3D model. The lookup tables are filled with entries describing the appearance of these features from all points of view about a model. Because pairs of vertices are not specific enough — any image pair can match any model pair by a suitable change of viewpoint — each match votes Hough-style for a <model, transform> pair, so that information from all indexing attempts can be accumulated. For this reason, the method does not scale well to complex scenes and large databases of complex models.

[1]The view sphere is an approximation valid under orthographic projection, in which each point on the sphere indicates one possible view of an object. When perspective projection is used, the abstraction no longer holds, since each point on such a sphere leads to multiple object appearances, which depend on the distance of the viewer from the object.

The work of Dickinson, *et al.* (1992a, 1992b), has similar motivations to our own, in that they wish to compute probabilities of association between non-invariant features observed in images and object models, however the approach they take is quite different. Their system is based on a small set of high-level volumetric primitives which sit atop a hierarchy of simpler features: the 3D "primitives" are composed of 2D aspects; aspects are composed of faces; and faces are composed of boundary groups. In contrast, our method uses simple groupings of features such as straight edge segments to directly index the 3D objects. These shapes are not constrained to arise from any given set of volumetric primitives.

The key to Dickinson, *et al.'s* method is that by using a fixed set of primitives, probabilities of association between features in any two layers of the hierarchy can be pre-computed into index tables. At run-time, probabilities for high-level hypotheses can then be automatically and rapidly generated using the tables, once low-level processing has determined the boundary groups. Models are represented as combinations of the primitives. Because primitive's dimensions and join positions are allowed to vary, while the number of primitives is small, the number of models that can be generated is quite large. Primitives then act like the neck of an hourglass through which associations must pass, from a large number of simple, low-level features, through a small group of primitives, on to a large set of complex models.

LUTs are built from many views using a tessellated sphere, which gives a roughly equal *a priori* probability for all viewing directions. The LUTs describe the conditional probabilities between any two levels of the hierarchy. The idea is to first index and recognize a set of primitives in the image, and then index actual models using combinations of the detected primitives. The probabilities are computed under the assumption that the set of views generated from the model database provide a good statistical sample. Conveniently, adding a new model to the database built from the old primitives does not change the lower-level hierarchy of probabilistic inferences from boundary groups up to primitives, so models may be included incrementally. Only the addition of a new low-level feature would require recomputation of those index tables.

The main difference with our approach is their use of a qualitative model representation.

While our system demands a fairly restrictive, quantitative specification of an object, theirs allows the dimensions of parts and part relationships to vary. This means that, in principle, they can recognize generic objects, for example, all coffee mugs, which they might represent as a straight circular cylinder (the cup body) joined to a circular cylinder with curved spine (the handle), while our system is more appropriate to particular instances of objects.

The drawback with their method is that the constraints provided by the use of a model are significantly weakened. Choosing such simple shapes as primitives (e.g. cylinders, cones), and in addition allowing their dimensions to vary, demands too much from segmentation algorithms. Noise will cause distortion and ambiguity among many of the boundary groups, aspects, and faces they use for indexing. Indeed, the only images they have tested their system with are synthetic ones. In contrast, using quantitative information from the model database means that such a complex hierarchical structure, with its series of less and less certain inferences, is unnecessary, and instead strong probabilistic inferences can be made directly from the image data to the model database.

2.3.3 Geometric Hashing

Geometric hashing (Lamdan & Wolfson 1988, Lamdan, Schwartz & Wolfson 1990) is closely associated with 3D-from-2D methods, because it has been applied mostly in that realm. It is more general than that, however, and is of a different flavor than the other invariant indexing schemes. This method is interesting and unique in that it pushes even more of the recognition process from the run-time into the pre-computation stage.

The features normally used are "interest points", such as corners and curvature extrema. For affine transformations, a basis can be created from three of the interest points. In this frame, the co-ordinates of all other points co-planar with the original three are invariant. The pre-processing stage consists of storing, for each set of interest points from the (planar) models, all other model points, in a hash table based on the coordinate values. At run-time, three interest points from the image are chosen as a basis, and the rest of the image points are hashed into

the index table. Each time there is a "hit" on a cell containing pre-stored points, a vote is recorded for all <model, basis> combinations stored there.

The interesting aspect of the method is that, to a large extent, the verification stage has been absorbed into the indexing stage. Every time we choose a basis and hash image points into the table, we are comparing the image against the "invariant templates" of all basis choices for all models at once. Choosing any correct match of three image to three model points should lead to a large vote tally and a strong indication for a particular model. The false positive count should be extremely low, because global information about a model is collected via the voting scheme. This means verification is largely moot, used more to establish an accurate pose than to reject incorrect hypotheses.

However, the efficiency of the method is only possible because each of the models is redundantly stored in the table (one time for each of many bases), the price tag being extremely heavy memory requirements. As well, noise degrades both the accuracy and the speed of the method since votes must be cast not just in the computed bin for each point, but in adjacent ones as well. Grimson & Huttenlocher (1990) notes that performance is poor even with small amounts of noise and clutter. This is due to the hash table becoming overloaded with entries, and indicates that the use of higher-dimensional spaces is important. There is also the problem of choosing an appropriate bin size. In fact, it is effectively impossible to find a single adequate size because the density of stored points varies throughout the index space. Additionally, grouping appears to be much less effective with this approach because the ideal basis is one in which the three points are widely separated (i.e., non-local) on the model (Rigoutsos 1992).

A final problem with geometric hashing is that, like most of the 3D-from-2D invariant methods, the features must be coplanar. Lamdan & Wolfson (1988) suggest two possibilities for handling more general 3D objects. One of these increases the time complexity substantially, requiring each image point to be hashed into multiple bins (similar to the idea of Jacobs (1991)); the other, which treats each view of a 3D object as a planar template, increases the already heavy memory requirements (similar to the tessellated sphere approaches). Without

	image type	clutter?	object type	database size
Oxford group	real	Y	flat	$\mathcal{O}(10)$
Geometric hashing	real	N	3D	$\mathcal{O}(1)$
Dickenson, et al.	synth	N	3D	$\mathcal{O}(10)$
Califano and Mohan	synth	N	2D	$\mathcal{O}(1)$
Clemens and Jacobs	real	Y	flat	$\mathcal{O}(10)$
Stein and Medioni	real	Y	2D	$\mathcal{O}(10)$
Beis and Lowe	real	Y	3D	$\mathcal{O}(10)$

Table 2.1: Comparison of indexing methods.

the planarity restriction, geometric hashing starts to look like the non-invariant methods, with their attendant problems.

2.4 How are Indexing Methods Evaluated?

Currently, there are no widely accepted benchmarks available either for object recognition in general, or indexing in particular. One reason is that current methods are still exploratory, and not yet at the level of an engineering discipline in which performance tolerances can be set. It is also due to certain practical difficulties: in acquiring models of the objects; in the number of parameters that affect performance; and in the large degree of variability of many of these parameters.

Obtaining models is problematic due to the time required to accurately measure the objects by hand. The tediousness of this process is one reason that most databases consist of at most ten objects (see Table 2.1).

Variables which complicate the comparison of indexing/recognition algorithms include:

(a) Model representation. Approaches vary according to choice of features, image data dimensionality (2D or 3D), and model feature dimensionality (2D or 3D). A method may exhibit very poor performance when applied to an object class favored by a different method.

(b) Lighting conditions. Indexing and recognition are strongly affected by the positions and intensities of light sources, due to the influence of specularities, shadowing, and contrast on feature detection. The range of possible lighting conditions is extremely large.

(c) Degree of occlusion. This variable has never been quantified for use in recognition experiments. One definition might be "the percentage of the image area of the target object that should be visible but is concealed". A more meaningful alternative for the indexing step would be "the percentage of features useful for indexing that are concealed". Unfortunately, this can only be determined post-recognition using the back-projected object, so it can't be measured if indexing fails. To date, more qualitative, descriptive measures have been used, for example "significant" or "mild" occlusion.

(d) Size of object in image. In general, the larger the object, the easier it is to recognize, because feature detectors give better results. A further consideration is the pose of the object with respect to the camera, since some viewpoints give rise to a greater number of features, or a more useful set of features, than others.

(e) Degree of clutter. A large number of distractors in an image not only slows down indexing, but increases segmentation and grouping errors, which then hampers indexing.

Because of these difficulties, researchers commonly provide indicative example images on which their algorithms have been successful, rather than providing a thorough coverage of all imaging situations. Typical unspoken assumptions are (i) that the object(s) occupies a large fraction of the scene; and (ii) that lighting conditions are not extreme (i.e., that a large fraction of the light is ambient, non-directional light). These assumptions are currently acceptable to the community because the problems of occlusion, clutter, and large database size are currently difficult enough that worrying about detecting tiny objects in the dark is not yet as issue. A demonstration of fast and accurate recognition under these two assumptions, for a large database in real, cluttered images, would be considered a very strong result.

In this work, there are two restrictions that apply. Firstly, there will be only one type

of primitive feature, the straight edge. This representation has many advantages: matching between 3D model segments and 2D image segments is straightforward; grouping relationships between segments are relatively easy to specify; and there exists a rich set of real-valued features that can be derived from sets of straight edges (such as various angles and edge length ratios). The disadvantage is that the class of objects that can be modelled is restricted to (non-convex) polyhedra. In defence of this choice, we note that there is currently no solution to the simplified version of the indexing problem implied by this choice of representation, that is, the problem of robustly indexing a large database of 3D polyhedra in images with occlusion and clutter, where the segmentation is computed automatically. A system that accomplishes this task will be an advance in the state of the art of object recognition. Furthermore, this indexing approach is easily extensible to more general features in the future.

The second restriction is that only rigid objects will be considered. This means that we cannot claim to recognize generic objects, such as "any coffee mug", but only specific objects (e.g. "Fred's coffee mug"). Generic objects are those defined by their component parts, and relationships between their components. Parameters which affect the specific object shape are allowed to vary. For example, a generic coffee mug might have a cylindrical body and a protruding loop for a handle, but the height-to-width ratio of the cylinder could change from mug to mug, as could the join positions of the handle to the mug body. Again, while it is desirable to solve the most general form of the recognition problem, restricting to rigid objects still leaves a complex unsolved problem. In fact, we argue later that our indexing method can handle objects with a small number of shape parameters, at the expense of increased memory usage.

Given these restrictions, it will be demonstrated that the approach to indexing proposed in this thesis is a strong competitor with other methods, while tackling a more difficult problem (3D-from-2D recognition) than most previous attempts. Our experimental approach will be to explore the properties of our technique using synthetic images, where ground truth is available. Scalability with respect to database size, resistance to image noise, and quality of hypothesis

generation will be quantitatively examined in this way. Because a complete quantitative investigation of performance with real image data is infeasible, we will instead present experiments which have both quantitative and qualitative aspects. We will loosely define three regimes of images for recognition, within which quantitative experiments will be performed. We believe this type of analysis provides enough information to enable a realistic comparison with any other indexing method.

Chapter 3

Indexing Without Invariants

3.1 Introduction

As we have seen, a great deal of work in indexing has concentrated on developing specific invariant features, which are appropriate for restricted classes of objects. Examples from the previous chapter include various types of constrained feature grouping: planar point sets, pairs of planar conics, sets of planar lines, etc. We believe it is important to move beyond these limitations to allow indexing to work for more general classes of objects. The fundamental premise of this thesis will be that

Premise 1: *Indexing must succeed even in cases where invariant features are unavailable.*

A second major limitation with many current methods is that the quality of a hypothesis is determined solely by the quality of the feature match, for example inversely proportional to the distance between model and image points in the feature space. The issue of how to weight hypotheses has recently received increased attention (e.g. (Ben-Arie 1990, Pathak & Camps 1993, Wheeler & Ikeuchi 1995, Shimshoni & Ponce 1995)), the basic approach being to estimate the probability that a given hypothesis is valid. In accordance with this trend, the second major postulate of this thesis is that

Premise 2: *Discriminating between strong and weak match hypotheses is key to providing efficient indexing for object recognition.*

In the past, *saliency* measures have been used for this task. Saliency refers to the ability of a feature (or set of features) to distinguish one object from another, i.e., it is a measure of the uniqueness of a particular feature within a model database. Turney, Mudge & Volz (1985) used a combination of match quality and feature saliency to weight their matches. They were able to pre-compute saliencies, but required a fixed set of 2D invariant features, thereby limiting its applicability for use with 3D objects. As well, a necessary assumption was that each feature be equally likely to appear in an image. This is unrealistic, and in fact our method contains machinery to explicitly determine these (variable) likelihoods. Nevertheless, the combination of match quality (closeness of match) and feature saliency (uniqueness of match) remains important for judging the quality of hypotheses. In the following section, we will see how these two measures can be embedded within a framework that also deals with the non-invariance of features.

3.2 Function Approximation and Probability Estimation

Given that we cannot rely on invariants, we are left with the task of representing the continuous set of appearances of a 3D object as a function of viewpoint. An interesting paper by Poggio and Girosi (1990) casts the problem of object recognition as that of *function approximation.* They show that, for simple wire-frame objects, it is possible to construct a function that will output "1" if an object is present in an image, and "0" otherwise, based on a small set of example images of the object. (Actually, the response rolls off smoothly from 1 to 0 as images become less and less "object-like".)

In their method, each image of an object generates a single feature vector $\vec{\mathbf{x}}$ in an

N-dimensional feature space. The recognition function for an object is given by

$$f(\vec{\mathbf{x}}) = \sum_{\alpha=1}^{K} c_\alpha G(\|\vec{\mathbf{x}} - \vec{\mathbf{t}}_\alpha\|) \tag{3.1}$$

where the $\vec{\mathbf{t}}_\alpha$ are training vectors extracted from example images of the object; $G(\|\cdot\|)$ is a radially symmetric basis function such as the Gaussian; and c_α are coefficients which depend only on the $\vec{\mathbf{t}}_\alpha$, and are determined by a simple matrix inversion in an offline training stage. Each $\vec{\mathbf{t}}_\alpha$ represents the shape of the object as it appears from one viewpoint, and the basis functions G are used to interpolate between these points. A hidden width parameter to these *Radial Basis Functions* (RBFs) determines the smoothness of the approximation. This approach is important because it demonstrates how a relatively small, finite set of examples can be used to represent the continuous appearance set of an object.

RBFs have been used extensively in the machine learning community for pattern classification, following the learning-from-examples paradigm. With only a slight modification from the above formulation, RBFs can be used to provide discrimination between several classes for which labelled sets of training vectors are available. Wan (1990), Ruck, Rogers, Kabrisky, Oxley & Suter (1990), and Hartman, Keeler & Kowalski (1990) have shown that RBF responses approximate *a posteriori probabilities* that a given test vector $\vec{\mathbf{x}}$ belongs to each of the classes. Used in this way, RBFs are similar to many other non-parametric pattern classification methods such as Parzen Windows and k-nearest neighbours.

We highlight the difference between the two closely related areas of function approximation and probability estimation because many recognition methods effectively use the former approach when the latter is more appropriate. Function approximation methods consider a set of data points $(\vec{\mathbf{x}}_i, y_i)$ sampled from an unknown function, and produce a function estimate which passes as closely as possible to those points. In the RBF approach, each input value is a feature vector representing an image shape and each output value is "1", which means the system is able to answer the question: "For a particular object, *is it possible* that the object corresponds to some image shape?"

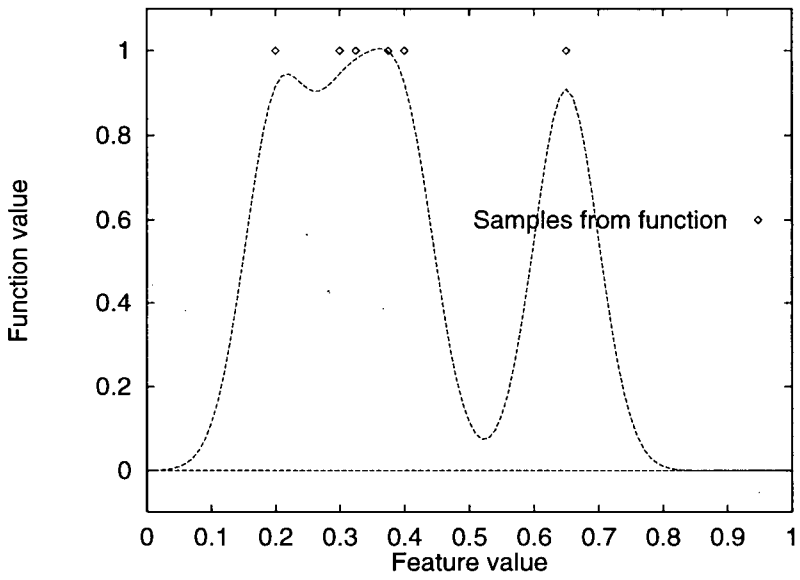


Figure 3.1: RBF function approximation.

On the other hand, probability estimation methods consider data from two or more classes, and try to estimate one function for each class. These are the density functions for each class, and if these are known, *a posteriori* probabilities can be determined and used for classification decisions. The type of question these systems can answer is: "For a particular object, *how likely is it* that the object corresponds to some image shape?"

Ben-Arie (1990) and Burns et al. (1993) investigate precisely this type of relationship, not for entire objects, but for very simple features: angles and distance ratios. They each present a detailed analysis which demonstrates that the values of the features are quite stable over a wide range of viewpoints. Malik & Whangbo (1997) presented a similar analysis for planar pairs of angles, with similar findings. Shimshoni & Ponce (1995) used the probabilistic peaking effect of angles to rank hypotheses. In this thesis, we are interested in the probability densities of more complicated 3D shapes for use in indexing.

A comparison between function approximation and probability estimation is given in Figures 3.1 and 3.2. Figure 3.1 shows an RBF function approximation in a 1D feature space.

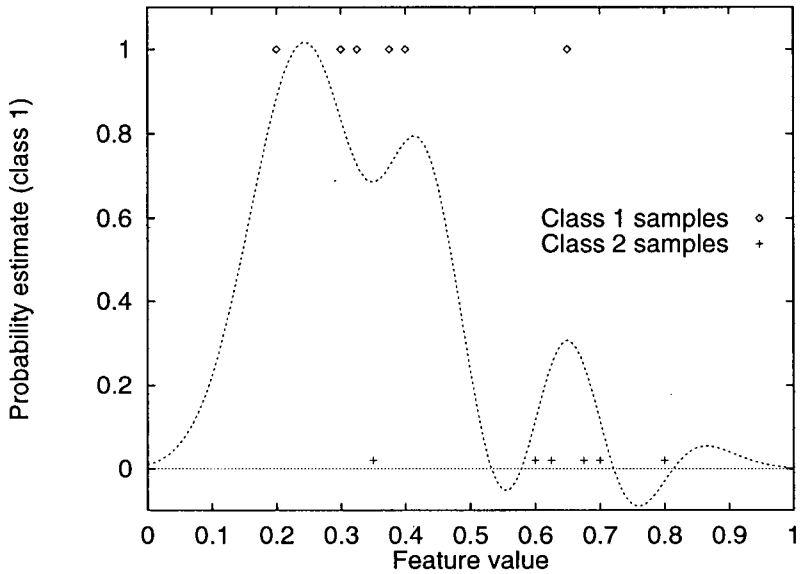


Figure 3.2: RBF probability estimation.

The output value for each sample is set to "1", and the plot shows that the function estimate very nearly passes through the full set of points, falling to zero for feature values far from any exemplars. In Figure 3.2, RBFs have been implemented for pattern classification. Samples from two different classes are shown, and the plotted curve is the density estimate for class 1. Output values for Class 1 are set to "1" and for class 2 to "0", and in regions of ambiguity the system trades off interpolation of both. (In the figure, class 2 data points are shown at height 0.02 for visibility.) Several "negative examples" from class 2 strongly influence the function on the right side of the plot, reducing the density in the area where only a single class 1 exemplar appears.

These figures demonstrate that the intuitive notions of match quality and feature saliency are aspects of the more formal property of probability. Referring to Figure 3.2, a test point with value $0.15 \leq x \leq 0.40$ has a high match quality (is close to some class 1 exemplars) and a high saliency (few non-class 1 exemplars are nearby), so it receives a high probability estimate. A point with good match quality but low saliency, such as $x = 0.65$ (the position of the isolated

class 1 exemplar), receives a lower probability, as does the converse, e.g. $x = 0.10$, which has high saliency but is further from any of the stored points.

In accordance with the above examination, in this thesis we adopt a probabilistic approach to indexing, seeking probability estimates for features sampled from images of database objects. An outline of the major steps involved is presented in Figure 3.3. This methodology addresses the two fundamental issues of the thesis — non-invariance of features and probabilistic ranking of hypotheses. As well, the use of smooth basis functions to interpolate between examples implies improved resistance to noise (**Problem 6** from Chapter 2). Small perturbations in the training or test data points should lead to graceful degradation of probability estimates, rather than catastrophic failure of the method. The approach is also compatible with real-valued features, which is important for **Problem 1** ("shape is continuous").

Several important choices remain, including which type of density estimate to use; which data structure is best to store and retrieve the training examples; and which features are appropriate for indexing. We begin with a discussion of probability estimators.

3.3 Estimating Probabilities

To index without invariants, we need to interpolate between the samples recovered from the index data structure, so that at any point in the data space we can determine *a posteriori* probability estimates that such a shape matches any given model. Omohundro (1989) describes several algorithms that can be used for this class of problems, which he calls "geometric learning tasks". One important distinction he makes is between "local" and "global" learning methods. This has to do with the "receptive fields" of the basis functions used in each method. The "units" in a back-propagation network, which are just basis functions by another name, are global because they can respond to inputs over the entire input space. This causes problems in the learning stage, which include extremely long training times and susceptibility to poor solutions, which are local minima of an energy function optimization.

Training stage:

STEP 1: Generate a set of training examples for each object by observing it from several points of view. These are $\vec{\mathbf{t}}_\alpha$ from Eqn. 3.1.

STEP 2: Store the associated feature vectors in an indexing lookup table.

STEP 3: Generate "indexing functions" such as RBFs from training vectors. This involves computing the c_α from Eqn. 3.1.

Recognition stage:

STEP 1: Derive feature vectors from image. These are $\vec{\mathbf{x}}$ from Eqn. 3.1.

STEP 2: Access the lookup table using $\vec{\mathbf{x}}$ to recover a set of nearest neighbour training vectors, indicating potential matches.

STEP 3: Use indexing functions to compute probabilities for each match hypothesis.

STEP 4: Rank hypotheses for verification according to probability estimates.

Figure 3.3: Indexing methodology.

On the other hand, Gaussians are (in practice) limited-support basis functions, and RBFs which use them correspond to the local methods. (For examples of RBFs with infinite support, see (Broomhead & Lowe 1988, Franke 1982).) Local methods are more appropriate for our problem, since the density for a class should fall to zero in regions of the input space that are far from any training examples. While a global method such as back-propagation might be able to tune the receptive fields of its units to become local given enough training data, it is more efficient to build this known constraint into the system from the beginning.

There are many methods of local, kernel-based classification which use sample sets drawn from each of the possible classes. In this thesis, we consider two simple, classic estimation methods, k-nearest neighbours (kNN) and Parzen Windows (PW), as well as a hybrid method which we call weighted kNN (WkNN), and the more complex RBF technique.

3.3.1 k-Nearest Neighbours

Using the notation of Duda & Hart (1973), the density for class m at point $\vec{\mathbf{x}}$ can be estimated as

$$p(\vec{\mathbf{x}}, m) = \frac{k_m/n}{V} \tag{3.2}$$

Here V is the size of a small volume, centered at $\vec{\mathbf{x}}$; k_m is the number of samples from class m enclosed by V; and n is the total number of samples in the space. Then, for C the total number of classes, a reasonable estimate for the *a posteriori* probability of class m is simply the fraction of the k samples enclosed by V that belong to class m:

$$p(m|\vec{\mathbf{x}}) = \frac{p(\vec{\mathbf{x}}, m)}{\sum_{c=1}^{C} p(\vec{\mathbf{x}}, c)} = \frac{k_m}{k} \tag{3.3}$$

The main advantage of this method is that it is fast: the neighbours can in general be efficiently recovered, and computing the densities from those is trivial. The disadvantages are that it may be less accurate than other methods, and that the choice of k may have a strong influence on the quality of estimates. While the importance of the method lies in how

it performs on real, finite data sets, (Duda & Hart 1973) does contain a short discussion of the theoretical aspects of convergence as $n \to \infty$.

3.3.2 Parzen Windows

In a similar vein, but now using a kernel form for k_m,

$$k_m = \sum_{i=1}^{N_m} \phi(\frac{\|\vec{\mathbf{x}} - \vec{\mathbf{x_i}}\|}{\sigma}) \tag{3.4}$$

where the $\vec{\mathbf{x_i}}$ are the training examples, and $\phi(\cdot)$ is a kernel, or window, function, with size controlled by the parameter σ. Inserting Eqn. 3.4 into Eqn. 3.2 gives:

$$p(\vec{\mathbf{x}}, m) = \frac{1}{n} \sum_{i=1}^{N_m} \frac{1}{V} \phi(\frac{\|\vec{\mathbf{x}} - \vec{\mathbf{x_i}}\|}{\sigma}) \tag{3.5}$$

Note that $\phi(\cdot)$ must be a legitimate density function in order for p to be also. Then,

$$p(m|\vec{\mathbf{x}}) = \frac{p(\vec{\mathbf{x}}, m)}{\sum_{c=1}^{C} p(\vec{\mathbf{x}}, c)} = \frac{\sum_{i=1}^{N_m} \phi(\frac{\|\vec{\mathbf{x}} - \vec{\mathbf{x_i}}\|}{\sigma})}{\sum_{i=1}^{N} \phi(\frac{\|\vec{\mathbf{x}} - \vec{\mathbf{x_i}}\|}{\sigma})} \tag{3.6}$$

Here N_m is the number of samples from class m and N is the total number total number of samples from all classes. If the window function $\phi(\cdot)$ is a unit hypersphere, Eqn. 3.6 essentially becomes a k-nearest neighbour estimate.

In this thesis, the Gaussian $G(\vec{\mathbf{x}}) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(\frac{\vec{\mathbf{x}} \cdot \vec{\mathbf{x}}}{2\sigma^2})} \equiv \phi(\frac{\|\vec{\mathbf{x}}\|}{\sigma})$ is used as the Parzen kernel. The potential advantage of this method over kNN is the use of the non-uniform kernel, which weights each training sample according to its proximity to the test sample. This provides a smoother and probably more accurate density estimate. The primary difficulty is to choose the window width σ (similar to the problem of choosing k in the kNN approach). A second problem is that each Parzen density computation includes all of the data points, even though most of them will fall outside the window. This is computationally expensive, and motivates the following method.

3.3.3 Weighted k-Nearest Neighbours (WkNN)

This technique is a cross between the previous two methods, and captures the presumed accuracy of Parzen windows and the efficiency of kNN. Probabilities are computed using a small number (k) of neighbours, weighted by a smooth window function:

$$p(m|\vec{\mathbf{x}}) = \frac{p(\vec{\mathbf{x}}, m)}{\sum_{c=1}^{C} p(\vec{\mathbf{x}}, c)} = \frac{\sum_{i=1}^{k_m} \phi(\frac{\|\vec{\mathbf{x}}-\vec{\mathbf{x}}_\mathbf{i}\|}{\sigma})}{\sum_{i=1}^{k} \phi(\frac{\|\vec{\mathbf{x}}-\vec{\mathbf{x}}_\mathbf{i}\|}{\sigma})} \tag{3.7}$$

This is a more local density estimate than the standard Parzen estimate, and faster to compute since only k samples are used. In high-density regions, it becomes equivalent to kNN, since all k neighbours will be close to the center of the kernel, where the response ≈ 1. In low-density regions, it becomes equivalent to the Parzen windows estimate, since only the few closest neighbours fall within the window, and there would be no effective contribution from the remaining training samples. Thus. assuming that PW is more accurate than kNN, WkNN only pays a price for its efficiency when the density becomes high, and even for low densities should do no worse than kNN.

3.3.4 Radial Basis Functions

Recall that Radial Basis Functions have the form

$$f_m(\vec{\mathbf{x}}) = \sum_{i=1}^{N} c_{im} G(\|\vec{\mathbf{x}} - \vec{\mathbf{x}}_\mathbf{i}\|) \tag{3.8}$$

where G are the radially symmetric basis functions (specifically Gaussians in our implementation); the $\vec{\mathbf{x}}_\mathbf{i}$ are the set of training examples; and the c_{im} are coefficients used to interpolate the values $f_m(\vec{\mathbf{x}}_\mathbf{i})$, which can be computed by inverting matrix G with elements $G_{ij} = G(\|\vec{\mathbf{x}}_\mathbf{i} - \vec{\mathbf{x}}_\mathbf{j}\|)$. By setting $f_m(\vec{\mathbf{x}}_\mathbf{i}) = 1$ for training examples $1 \leq i \leq N_m$, and to zero for $N_m < i \leq N$, we obtain probability estimates

$$p(m|\vec{\mathbf{x}}) = f_m(\vec{\mathbf{x}}) \tag{3.9}$$

which approximate the Bayes optimal discriminant function (Ruck et al. 1990).

RBFs have a similar form to Parzen windows, but with more free parameters (the c_{im}). The advantage is the potential for better estimates if the parameters can be properly set. The disadvantages are extra computation and the extra memory to store the c_{im}. Furthermore, RBFs are designed for interpolation, and while they can be used for probability estimation, it is troublesome that the resulting functions are not constrained to lie in the range $[0, 1]$ - see Figure 3.2. This contrasts with the previous three methods, whose *a posteriori* estimates each provide a "partition of unity" over the classes, at each point in the input space.

3.3.5 Comparing Estimation Methods

What will make one method a better choice than any of the others? The three main issues are accuracy, speed, and space requirements. Regarding accuracy, a greater number of parameters for a method means that it can model a more complex density function with a smaller number of basis functions, assuming there is sufficient training data to compute reasonable parameter estimates. If the number of modes in the underlying distribution is not large, or the amount of available training data is not large, it is expected that the simpler approaches will work as well or better than the more complex. Accuracy is the most important consideration here because having good estimates significantly reduces the number of expensive verifications. In this work, we will not compare estimation methods by testing performance on simple, analytical functions. Rather, we assume that indexing performance — how well each method ranks hypotheses using an actual model database — is the best measure of accuracy.

At runtime, speed is crucial, and RBFs, for example, require more computation than k-nearest neighbours. However, this cost is relatively small compared to the cost of doing extra verification, so concerns about computation time for the probability estimates are overwhelmed by the requirement for accuracy of hypothesis ranking. Finally, with respect to space requirements, RBFs use more memory than the other methods, to store the basis function coefficients, but this is small relative to the space occupied by the training examples. Thus there is only a small advantage for the simpler methods like kNN.

3.4 Index Structure

Our core algorithm requires that matches be efficiently recovered from the pre-stored data points. To account for the continuous nature of the data, the finite number of stored samples, and for noise, we must not look for exact matches in the index table, but instead for the closest matches that can be found, the "nearest neighbours". Here, speed is essential for good runtime performance, and the obvious choice is to use a multi-dimensional array, or lookup table (LUT). This would appear to have the greatest potential for efficiency, as the access time for any cell is $\mathcal{O}(1)$.

For many types of data this approach works well. However, due to the nature of the index space — each dimension corresponds to a continuous feature — a bin size must be chosen for each dimension and the space divided into regular cells. Then, to be assured of finding nearest neighbours, there are two possible approaches, neither of which is efficient in high-dimensional spaces. The first approach retains the constant time lookup, but does so by storing each of the original data points not only in the bin corresponding to its coordinates, but in all adjacent bins as well. This redundancy is exponential in the dimension of the index space, and leads to impractical memory requirements.

The second approach consists of a runtime search of the same exponential set of adjacent bins. For larger dimensionalities, this eliminates the speed advantage of lookup tables over other data structures (see Chapter 5). The choice of bin size is also a complex issue, since a choice which is too small will mean a search of many more bins than necessary, while a choice which is too large will return many points that are far from the query, and will require a further linear search to extract the closest neighbours. In fact, due to non-uniform densities throughout the space, there is no perfect choice.

In the worst case, we would need to do an exhaustive, linear-time search through the stored points. Several alternative data structures that provide more efficient retrieval in Euclidean data spaces are available (see (Omohundro 1989)). One of the most versatile and efficient

kd-tree construction:

STEP 1: Initialize with the complete set of data in the root node.

STEP 2: Determine the variance of the data along each dimension of the space.

STEP 3: Choose the dimension d with the largest variance, and split the data at the median value m in that dimension, so that half the data falls on either side of the cut plane.

STEP 4: Form two child nodes to represent the two subsets of data, and in the parent node store d and m.

STEP 5: Iterate steps 2 to 4 with each of the child nodes, until each node contains $\leq l$ data points, l small. These are the leaf nodes.

Figure 3.4: kd-tree construction.

of these is the well-known kd-tree data structure (Friedman, Bentley & Finkel 1977). This is a balanced binary tree that adaptively partitions the space according to the local density of data, placing more cells in dense regions and fewer in sparse areas of the space. Under certain assumptions, lookup time is $\mathcal{O}(\log N)$ in the number of stored points.

Figure 3.4 gives the steps involved in building a kd-tree. Once the tree has been built, there is an efficient branch-and-bound search algorithm that can be used to recover nearest neighbours. First, the tree is traversed to find the cell containing the query point. This is done by testing the query point against the cut plane at each level, keeping to the appropriate side on the way down. The data point found in the leaf cell (we have assumed $l = 1$) is a good estimate for the closest point, but it is possible that a closer match lies just beyond one of the cut planes. An efficient backtracking policy is able to prune whole subtrees by checking whether

the closest possible point in the subtree is further than the current nearest neighbour estimate.

In the worst case of data scattered randomly throughout a high-dimensional space, a large fraction of the stored data may have to be examined to determine the exact nearest neighbours (Sproull 1991), but in practice this method has proven to work extremely well for low to moderate dimensions. In the object recognition problem, the dimensionality of index space is practically limited by the robustness of feature detection algorithms, so the problems of high-dimensional spaces may be mitigated. Furthermore, data will be clustered rather than random, because the number of degrees-of-freedom giving rise to the data — two view angles — will in general be smaller than the dimensionality of the index space, meaning that samples from any single grouping will fall on a 2D manifold. (This is not exactly true, since noise perturbs the samples away from the ideal manifold, but the disturbances should be small.) The trade-offs between nearest neighbour recovery by table lookup versus kd-trees will be quantitatively explored in Chapter 5. It will be shown that kd-trees are much more efficient than either LUTs or exhaustive search for moderate to high dimensionalities, and that a form of approximate NN search exists which provides an even larger speedup, at the cost of occasionally settling for a non-optimal set of neighbours.

A second consideration for the indexing data structure is the space requirement. The kd-tree data structure is less efficient than hashing by a factor of two, requiring N internal nodes in addition to the N leaf nodes. We can estimate the minimal amount of space required for one model, such that view-sphere coverage is complete. Full coverage for a model will be obtained when there are enough training views so that most shapes do not change much between one view and each of the nearby, surrounding views. This can be quantified using the feature vectors corresponding to the shapes, and noise estimates for each feature dimension. When the views are close enough so that feature vectors describing the same model part do not differ by more than the expected noise level in each dimension, then the coverage can be considered dense. This number of views is order-of-magnitude 100.

For each training sample, there is a minimal amount of information that must be stored,

which is some number of bytes for the feature vector (N_f) and some for a unique label indicating the underlying feature grouping (N_l). The feature vector will require $N_f = 2I$ bytes, where I is the dimension of the index space, and we have assumed that each dimension can be divided into $2^{16} = 65536$ values without losing much power of discrimination. (In fact, 1 byte or 256 values will often be sufficient.) The label will require $N_l = \lg(total\#groups) = \lg(Nmodels \times \frac{Ngroups}{model})$ bits. For a 1000 model database with 100 groupings per model, this gives $N_l \approx 17$ bits ≈ 3 bytes. The total space requirement for one data sample in a 10-dimensional feature space is then $N_f + N_l = 2I + N_l = 2 \cdot 10 + 3 = 23$ bytes. The total space requirement for a 1000 model database is thus 1000 models $\times$100 views per model $\times$100 groupings per view $\times$23 bytes per grouping = 230 Megabytes. This is not unreasonable by today's standards, and means that we can focus more on the important question of how to efficiently retrieve neighbours from a database with 10^7 entries.

A further memory issue is that the more data available, the better the probability estimates will be. Better estimates mean more accurate hypothesis ranking and faster recognition. However, as already mentioned, increasing the amount of data engenders the twin problems of increased space requirements and increased lookup time. An important issue that will be covered in Chapter 6 is *data clustering*, whereby several data points may be coalesced into a single point, without significantly affecting the accuracy of the probability estimates.

3.5 The Choice of Features

The choice of features for object recognition is often a trade-off between stability and descriptive power. In general, simpler features are more robustly detectable, but less able to adequately represent a wide class of objects. In this thesis, we have chosen the straight edge as the most primitive feature, the building block from which all shape descriptors will be formed. While feature detection has been an active area of research for decades, the simple straight edge remains the most robustly detectable and widely used of all features. Edges are largely invariant

to rigid transformations and lighting conditions, the two predominant causes of variability in object appearance. Although single edges are not salient features, *groups* of edges are a rich source of complex shapes that can be used for indexing.

There is one constraint on the choice of features that is imposed by the requirements of indexing, which affects the minimal grouping size. Feature vectors must embody enough specificity to reduce the number of potential matches with the model database from the complete set of all possible matches. This will only be true if the number of features is at least one greater than the number required to completely specify the model's rigid transformation.

Three points are sufficient to fully constrain a rigid transformation. This implies a minimum of two (non-collinear) segments. However, any three points in 3D can be rigidly rotated and translated so that their projection aligns with any three-point image configuration. This means that a feature vector generated from such a shape is entirely *non-specific*, and at least one more point must be added to make the group useful for indexing. In terms of edges, at least two edges that don't share any endpoints are required, or three in case segments coterminate.

The more complex a set of features, of course, the more unique the shape and the more specific the shape descriptor will be. In other words, the more edges that can be reliably collected together, the more accurate the indexing will be and the fewer false positives to be removed by the verification stage. In Chapter 4 we show that even small sets of 3 or 4 edges are sufficient to discriminate within a small, 10-model database.

3.5.1 Perceptual Grouping

A brute force grouping algorithm would form all possible combinations of segments of a given cardinality g. For an image with N segments there are $\binom{N}{g}$ such feature sets, each of which would be a potential structure for matching with the model database. This large number of queries would overload even an extremely efficient indexing method (Clemens & Jacobs 1991). *Perceptual grouping*, on the other hand, is a good way to trade off completeness for an

improvement in time complexity, without compromising performance.

Perceptual grouping methods (Witkin & Tenenbaum 1983, Lowe 1985, Biederman 1985, Saund 1993) use non-accidental properties of images to collect features that are likely to come from the same object. Witkin & Tenenbaum (1983) argue convincingly that groupings which satisfy non-accidental properties are parts of images that deserve further investigation. Some of these properties are: parallelism of curves; collinearity of points or other features; coterminating curves; and symmetric curves. Because such configurations are not likely to arise by chance in an image, these 2-dimensional clusters can be used to make probabilistic inferences (Lowe 1985) about the 3D structure of the world. Conversely, if these properties hold in the 3D world, then they hold in the projected 2D world for all viewing directions. This has been dubbed the "viewpoint consistency constraint" (Lowe 1987b).

One important aspect of all perceptual groupings is *proximity*, which also satisfies the viewpoint consistency constraint. In fact, proximity is a strong enough property to be used for grouping, even in the absence of further perceptual cues. The significance of the previously mentioned perceptual properties depends strongly on the proximity of the component features with respect to the background density of similar features (Lowe 1985). All groupings in this thesis will therefore be local in nature, satisfying the proximity property, and are deemed "Local Feature Sets" (LFS).

Grouping in itself, even without indexing, can greatly enhance search efficiency by restricting the number of feature sets requiring investigation. The extent of the savings can be estimated by comparing the number of LFS to the total number of image feature sets $\binom{N}{g}$. Consider, for example, segment chain groupings, and assume that the maximum number of coterminations at one end of a segment is 2. (It will likely be greater than 2, but the *average* number should actually be close to 1.) Then the number of chain groupings of cardinality g is $\leq N \times 2^{(g-1)}$. For fixed grouping size g, this reduces the number of image groupings from $\binom{\mathcal{O}(N}{g)=\mathcal{O}(N^g)}$ to $\mathcal{O}(N2^{(g-1)}) = \mathcal{O}(N)$.

Grouping can be combined with indexing to further reduce the search complexity. While

grouping restricts the *image space* search, by choosing a subset of interesting image groupings; indexing restricts the *model space* search: for each image grouping, a small subset of model groupings is recovered from the database. Lowe (1985) notes: "...to the extent that relations formed by perceptual organization are stable under different viewpoints and imaging conditions, they can be used as reliable index terms..." (p. 20). Using perceptual grouping to restrict attention to certain types of feature set does mean that relevant groupings may be overlooked. However, the huge size of the search space makes such a trade-off unavoidable.

3.5.2 Specificity, Robustness, and Redundancy

The *size* of a grouping is defined here as the dimensionality of the feature vector generated by the grouping. Consider a chain of N coterminating segments. The feature space for this grouping type is defined by the $N-1$ angles between consecutive segments, and the $N-1$ edge length ratios, giving a feature space dimensionality of $2N-2$. Note that once the canonical mapping between a grouping type and a feature space has been set, there is a one-to-one correspondence between a "feature grouping" and a "feature vector", and these two terms will be used interchangeably.

Larger feature groupings are more desirable because their shapes are more specific. Unfortunately, for each pair of object features that share some 3D property, such as a coterimination, there is some probability that once an image is acquired, the corresponding image features will not exhibit the same property. This may be due to feature detection, occlusion, or lighting conditions. As grouping size increases, this problem is compounded, and it becomes more and more likely that the grouping as a whole will not be detected. Thus, there is a practical trade-off between the *specificity* and the *robustness* of a grouping that is a function of the grouping size.

The solution is to build redundancy into the system. For example, if the 4-segment chain grouping $ABCD$ is to be stored in the index, the 3-segment chain sub-groups ABC and BCD should also be stored, so that indexing can succeed in spite of the possible failure of feature

detection or grouping for features A or D. (Which feature subsets remain valid perceptual groupings is specific to the original grouping type. In this case, removing a link from either end of a segment chain leaves a smaller chain, but removing an inner link destroys the nature of the grouping.)

At runtime, all sizes of grouping should be sought in the image, with grouping and indexing procedures applied in parallel for each separate grouping type. A large image grouping should generate a higher probability than any of the redundant sub-groupings, due to its greater intrinsic specificity. In case parallelism is unavailable, an "optimal" grouping size can be found by experimentation. This will depend on the time available for grouping, edge detector performance characteristics, etc. Because in the indexing stage false negatives are worse than false positives (which can be detected and removed in the verification stage), the "optimal" experimental grouping size may be smaller than is ideal for specificity.

3.5.3 Occlusion and Redundancy

It is always possible that any single feature grouping may be obscured from view by an occluding object. It is therefore important that any (unobstructed) view of an object contain several Local Feature Sets. This is a redundancy of grouping *location* rather than of grouping *size*, and it means that indexing may succeed even if only a small portion of an object is visible. The use of local groupings dovetails with the nature of occlusion, in which a cluster of features in a local region is obscured while other regions are left fully intact.

3.5.4 Partial Invariance

Although it is important not to rely on fully invariant features for indexing, any invariances that do exist vastly reduce the amount of training data required. In 2D, angles and edge length ratios are invariant. These are natural choices for features, rather than absolute lengths or endpoint locations, whose variation with image position must be modelled. In 3D, angles and

relative edge lengths are invariant to translation, scaling, and image plane rotations[1], but not to out-of-plane rotations. This partial invariance means it is only necessary to model the two degrees of freedom of the shape descriptors, instead of the full six DOF of the set of possible object transformations. Thus, making features "as invariant as possible" should always be a consideration when creating the canonical shape-to-feature-space mapping for any grouping type.

3.5.5 Parametrized Objects

It is now possible to consider how one of our original restrictions might be relaxed, that is, that only rigid objects can be handled. Suppose instead that there is a database object that contains a single articulation, a hinged part. This adds an extra degree of freedom to the shape variation. Instead of acquiring one set of training views over the view sphere, we could require one of these sets for each of several values of the extra model parameter, the articulation angle. Then the interpolation and probability estimation would be over the larger set of samples from a 3 DOF system. In this thesis, we have not attempted to model any parametrized objects.

3.5.6 Summary

Redundant Local Feature Sets are used to overcome noise and occlusion problems. Larger groupings provide specificity while smaller groupings provide robustness. Many types of grouping are possible, each having its own canonical mapping to feature space, and many different types should be used to provide a rich representation. Feature groupings are the fundamental shape fragments for which probability estimates are formed and match hypotheses generated. Shape descriptors derived from feature groupings should be as invariant as possible, to reduce the dimensionality of the manifold that must be modelled.

[1]This is true for weak perspective, and close to being true for full perspective projection in the vast majority of cases.

3.6 Improving Probability Estimates

3.6.1 Incremental Learning

An important advantage of the kernel-based approaches to learning-from-examples is the ability to update a mapping simply by adding new training examples. This is particularly useful for the density estimates in our method, which are initially generated from synthetic models under certain assumptions, and may not closely resemble the true distribution of object features as they appear in real images. Acquiring training data for the estimates will therefore take place in two stages, which we have called the "bootstrap" and "incremental" learning stages. The initial, bootstrap stage uses hand-input, 3D CAD-like models to generate training vectors. In the absence of other information, the following assumptions are used[2]:

(a) Equal *a priori* probability that any model appears in an image.

(b) Equal *a priori* probability for any viewpoint of a given model.

(c) Edge data simulated from the CAD models provide a good facsimile to edges detected in real images. (In fact, if this is not mostly true, the verification stage will fail.)

Following assumptions (a) and (b), training images will be formed by projecting each model from a set of evenly-spaced positions about the view sphere. The exact tessellation used to define these positions is unimportant so long as the views are fairly closely spaced to cover all aspects of the object.

What is learned in this bootstrap stage can be analyzed according to Bayes' Theorem:

$$p(m|\vec{\mathbf{x}}) = \frac{p(\vec{\mathbf{x}}|m)P(m)}{p(\vec{\mathbf{x}})} \tag{3.10}$$

The prior $P(m)$ falls out from assumptions (a) and (b), and is estimated as $\frac{N_m}{N}$, where N_m is the number of samples of grouping m, and N is the total for all groupings. $p(\vec{\mathbf{x}}|m)$, the probability

[2]In fact, there is another assumption used in this stage of training, which is that extreme perspective distortion is anomalous. Bootstrap training takes place in the weak perspective regime. For recognition to succeed in cases where the object is perspectively distorted, at least one feature set must be recognizable from its undistorted state. Incorporating image data over time from successful recognitions will gradually extend the range over which the object may be indexed, to eventually include strong perspective images.

distribution for grouping m, arises from sampling the view sphere according to assumptions (b) and (c). $p(\vec{\mathbf{x}})$ is simply the normalization $\sum_{m=1}^{M} p(\vec{\mathbf{x}}|m)P(m)$.

Once bootstrap learning is completed, recognition performance will depend on the degree of view sphere coverage provided by the training set, and the degree to which assumptions (a-c) are correct. If the system is able to identify an object in an image, the bootstrap estimate can be refined by adding to the index those image groupings determined to be in correspondence with the recognized object. This is what we call "incremental learning".

Adding a new example pair $\langle \vec{\mathbf{x}}, m \rangle$ increases $P(m)$ relative to other $P(m')$, relaxing assumptions (a) and (b). It also adjusts $p(\vec{\mathbf{x}}|m)$ in the direction of those feature vectors useful for recognition (i.e., those detectable in real images), relaxing assumptions (b) and (c). Provided enough images are available, probabilities should approach the "true" underlying distributions for each grouping.

A third, intermediate learning stage could be used in case assumption (c) is not a good one, which occurs when synthetic and real-image distributions are significantly different. One example would be a surface marking that is specified on a model (so it appears in all synthetic images), but on the real object has a very small contrast with the surrounding surface (and in reality often fails to be detected). An intermediate learning stage could be used to bridge the gap between the types of feature groupings generated in the synthetic, bootstrap phase and those detectable in real, unconstrained images, by applying incremental learning in a controlled environment.

A controlled imaging environment would be one in which the object is in isolation, covers a large portion of the image, and is free from perspective distortion. Lighting would be as close as possible to the imaging situation expected in practice or, if this were variable or unknown, non-directional lighting could be used. A synthetically trained indexing system could succeed on these special images, and new training vectors could be extracted and used to update the database. One can imagine iterating this stage a few times for more and more difficult lighting conditions, object sizes, etc., to further mold the density estimates before

moving to unconstrained and cluttered scenes.

This type of learning stage would also reduce reliance on assumption (b) if, for example, the object had only one natural resting position, and there was a natural relationship between camera and object, e.g. the camera was always at the same height relative to the object. In the latter case, the number of degrees of freedom of the system would be effectively reduced from two to one, and the typical 2D manifold of feature points in index space would collapse to a 1D curve.

3.6.2 Background Distribution

As described to this point, our method generates accurate probability estimates only if the database provides a sample set that is representative of the rest of the world. In general, we will need to determine $P(clutter)$ and $P(\vec{\mathbf{x}}|clutter)$, where *clutter* encompasses all image groupings *not* accounted for by a known object. Given unlimited computing resources, this could be accomplished with the incremental learning procedure by adding *unrecognized* rather than recognized groupings to the index. This would give the most accurate estimate of the background distribution, but would also lead to massive use of memory.

It should be possible to represent this huge amount of data more compactly. Two simple approaches are: (i) as a constant level over the entire space: $P(clutter|\vec{\mathbf{x}}) = K$; or (ii) represented in the same way as the model distributions, as a set of samples, but at a much reduced density, keeping the number of prototypes to a manageable level over the entire index space. A slowly-varying *clutter* distribution could be produced using kernel estimators with large window functions.

The first case, a constant level background, is not useful when indexing with a single grouping type. This is because adding a constant effectively scales (down) *a posteriori* estimates, which does not change relative probability levels. However, it would be important when combining results across grouping types, since different background levels would scale estimates differently for each grouping type. The value for a constant level can be estimated as follows.

Suppose that *clutter* is evenly distributed throughout the *occupied* portion of index space, and zero elsewhere. The occupied portion of index space V_{occ} can be determined by dividing the space into bins, taking samples of clutter from real images, and discarding those bins which are empty. In higher-dimensional spaces, a large fraction of the volume may be vacant, with some regions corresponding to non-physical situations, e.g. self-intersecting configurations of segments.

Let the average fraction of image groupings that correspond to models and clutter be f_m and f_c, respectively, with $f_m + f_c = 1$. The expected number of clutter points per unit volume in index space is

$$\langle C \rangle_v = \frac{M \times \frac{f_c}{f_m}}{V_{occ}} \tag{3.11}$$

where M is the total number of stored model feature vectors. To include clutter in the kNN estimate, for example, we need to determine the effective number of clutter neighbours k_c that exist in the volume implied by the k exemplars recovered from the original NN search. If we assume the volume V_m occupied by the k neighbours is spherical, with radius equal to the distance between the query point and the k^{th} neighbour, then

$$k_c = \langle C \rangle_v \times V_m, \tag{3.12}$$

which will not, in general, be integer-valued. The *a posteriori* probabilities can then be computed as

$$p(m|\vec{\mathbf{x}}) = \frac{k_m}{k + k_c} \tag{3.13}$$

for model grouping m, and

$$p(clutter|\vec{\mathbf{x}}) = \frac{k_c}{k + k_c} \tag{3.14}$$

for clutter. Estimates for Parzen, WkNN, and RBF methods can be computed using similar strategies.

The second case, a slowly varying background level, can be generated in a manner similar to the densities of the models, by storing a number of samples. Using a lower density of

prototypes means averaging over a larger volume, and implies that higher-frequency variations will be lost. It also means that a separate kd-tree is required for NN lookup, since the closest clutter neighbours to a query are likely to be far away relative to samples from the model database. To estimate clutter density within the k-nearest neighbour approach, we can first recover from the clutter index a set of k clutter neighbours. If V_c is the volume occupied by the k nearest clutter neighbours, computed in the same manner as V_m above, then the effective number of clutter neighbours contained in V_m is

$$k_c = k \times \frac{V_m}{V_c} \tag{3.15}$$

and $p(m|\vec{\mathbf{x}})$ and $p(clutter|\vec{\mathbf{x}})$ are as given in 3.13 and 3.14.

A nice way to generate a useful set of representative prototypes for the clutter would be to first generate a very large set of clutter samples, and then to reduce the number by using a *clustering* method. Clustering is discussed in Chapter 6.

3.6.3 Dimensionality Weighting

Because the notion of distance is so critical in classification and density estimation, with "nearby" training samples strongly influencing results and "distant" samples not at all, setting the relative weights of the dimensions is an extremely important task. In many learning problems, it is not immediately obvious which are the more important features. Variables may be measured in different units, making it difficult to set the relative scales so that each feature has an optimal amount of predictive influence. A wrong choice can lead to very poor system performance, as noted by Lowe (1995) for classification problems and Botros & Atkeson (1991) for function approximation.

For some combinations of features, it is possible to set reasonable values by hand, using expected noise levels and knowledge of what should be considered "similar". Alternatively, it may be possible to optimize a measure such as classification performance with respect to the dimensionality weightings. Lowe's (1995) Variable-Kernel Similarity Metric (VSM) Learning

method takes this approach.

VSM adopts a particular form of smooth kernel for interpolating between examples. The kernel is *variable* in that its size is proportional to the average distance to the first v neighbours. The kernel is differentiated to determine, for each sample, the change in cross-validation error due to changes in dimension weights. Optimal weights are computed by iterating over the samples and incrementing the weights at each step to reduce the error. Conjugate gradient descent is chosen as the optimization method, due to its good convergence properties. The small number of parameters to estimate (feature space dimension + 1) implies a much faster training period than alternatives such as back-propagation. It also means that a large amount of data is included in the estimate of each parameter (so, for example, overfitting is unlikely). A modified version of VSM is used in the subsequent chapter to check the initial, by-hand settings of each grouping type.

3.7 Summary

In this chapter we have laid out an approach to indexing that solves the six problems of Object Recognition described in Chapter 2, and which focuses on our two major premises concerning indexing stated at the beginning of this chapter.

- **Problem 1**, the fact that shape is continuous, is dealt with by the use of kd-trees, instead of multi-dimensional arrays or hash tables. The latter are more appropriate for discrete rather than for real-valued data, and in low-dimensional spaces.

- **Problems 2, 3**, the ambiguity of image shape data, and **Premise 2**, that discriminating between strong and weak hypotheses is important for efficient indexing, are dealt with by the use of probability estimates.

- **Problem 4**, the variation of shape with viewpoint, and **Premise 1**, the variation of shape descriptors with viewpoint, are dealt with by sampling feature groupings over many view-

points and interpolating between the samples using indexing functions (i.e., the density estimates).

- **Problem 5**, the choice of shape descriptor, is dealt with by the use of Local Feature Sets which, in addition to being local and redundant, are partially invariant.
- **Problem 6**, that of noisy data, is dealt with by the use of redundancy of Local Feature Sets (to counter occlusion and undetected features), and by the use of smooth probability distributions (to withstand minor distortions in feature values).

Chapter 4

A Complete Object Recognition System

The approach to indexing described in the previous chapter is designed specifically with the needs of an object recognition system in mind: the type of data to be stored in the index (real-valued, multi-dimensional points), the use to which the recovered data will be put (generating match hypotheses for verification, hence the calculation of probabilities is important), etc. The technique is therefore relevant mainly within the context of object recognition as a whole, and to the extent that it can be shown to work in practice.

In this chapter, the details of a complete recognition system, in which the indexing method has been embedded, will be presented. The system contains all of the elements crucial to any model-based recognition system, including the model database itself (and choice of model representation), feature detection, feature grouping, indexing, and matching (hypothesis verification). It is useful to keep in mind that the choices for each of these are not unique, and that the ability of the indexing system to generate valid hypotheses is not independent of any of them. However, demonstrating a degree of success in at least one particular domain is important, to show that an approach does not simply provide a theoretically elegant solution to a problem as it has been abstracted, but applies directly to the real-world version of that

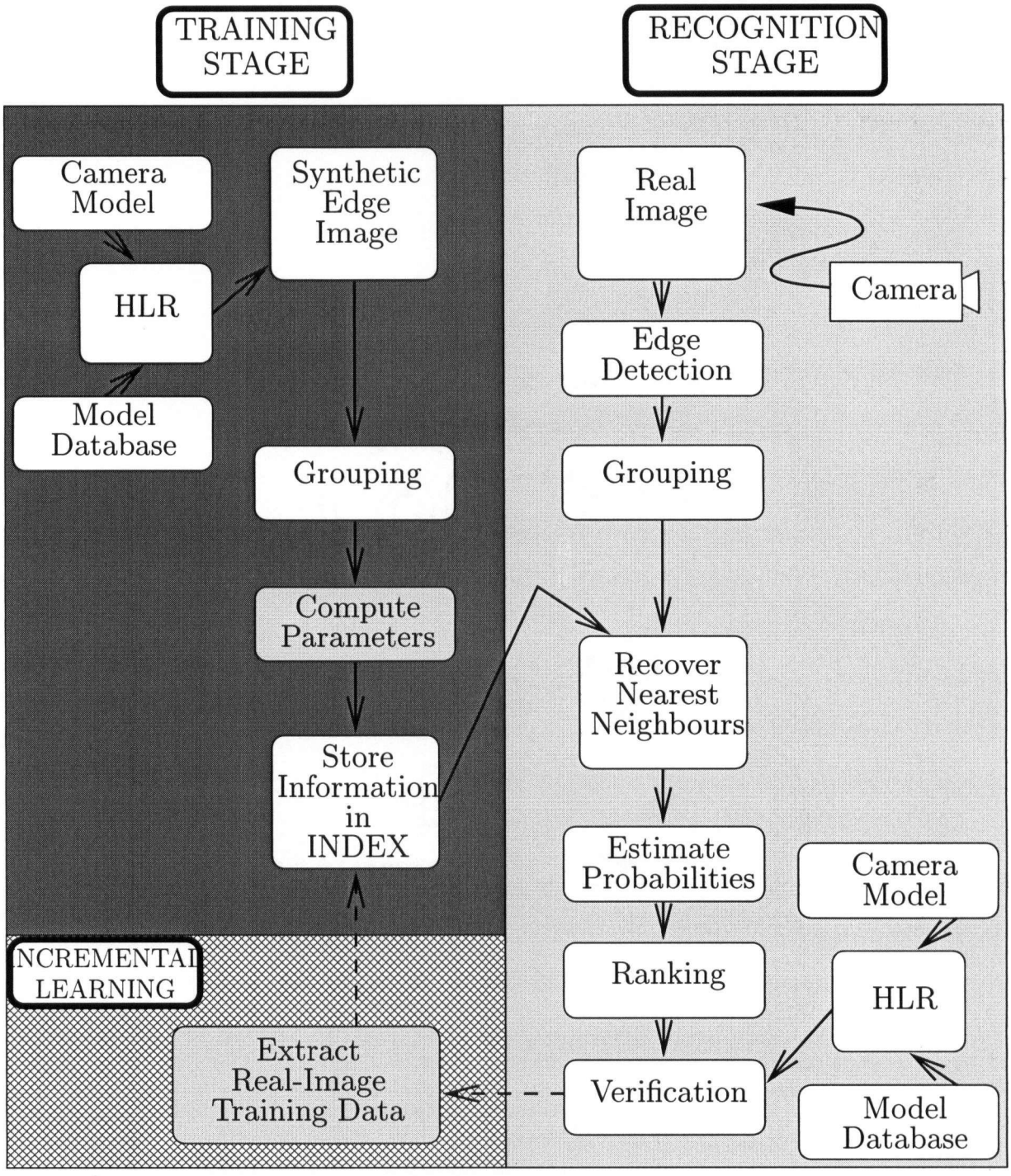


Figure 4.1: Recognition system overview. Shaded components are not required in all versions of the system. (HLR ≡ hidden-line removal.)

problem for which it was originally designed.

Figure 4.1 gives an overview of the entire recognition system. It divides cleanly into two main components, an offline training stage, and a runtime recognition stage. An additional "feedback" module is included for incrementally updating the index using real image data. I will discuss each component in turn, starting with the training stage, and noting that there are modules used by both main stages which will be dealt with when they first appear.

4.1 Synthetic Image Generation

4.1.1 Model Database

Models are polyhedral and are stored as lists of 3D points, edges, and faces. Polygonal surface markings are also included in the representation, and these are lists of 3D points and edges, with a label to indicate the surrounding face. Face information is used in a hidden-line removal algorithm that makes model matching more accurate than with transparent, wire-frame models.

Models are currently measured by hand (although see Chapter 7), in an object-centered coordinate system. This is a time-consuming process, and for this reason the model database available for experiments remains small, about 10 objects. To our knowledge, most other polyhedral object databases consist of abstract geometric figures rather than real-world objects, although a few of our models were recovered from a remote source. While no images of these objects were available for testing, they have been used as distractors within the index.

4.1.2 Camera Model

A pinhole-camera model of full perspective projection is used (Figure 4.2). The coordinate system is left-handed; the u-v image plane is aligned with the camera x-y plane and both have the same origin; f is the focal length; the zero of the z axis is at the focal point; and for an object at world point (x, y, z), there is a corresponding image point (u, v) where $u = \frac{-fx}{z}$ and $v = \frac{-fy}{z}$.

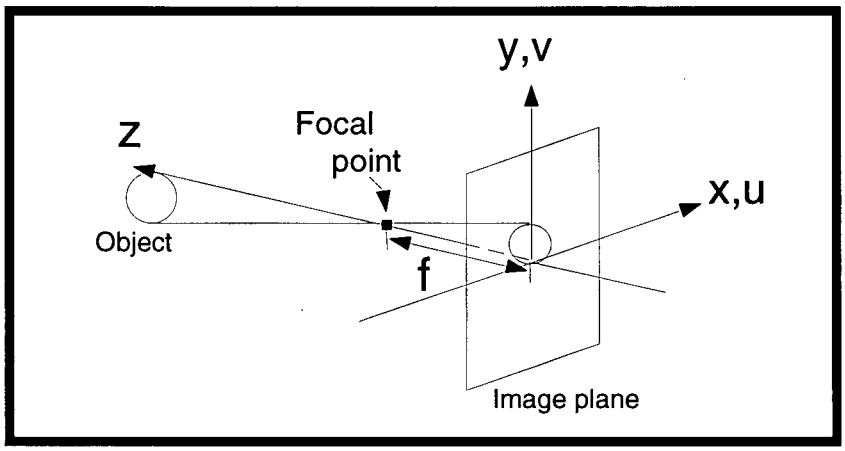


Figure 4.2: Camera model.

The above model is sufficient for generating synthetic images of database objects, but in order to process real images, camera calibration is required. In this thesis, we have used the simplest form of the Tsai method (Tsai 1987), in which

$$u = C_x + \frac{s_x}{D_x}(\frac{-fx}{z}) \tag{4.1}$$

$$v = C_y + \frac{1}{D_y}(\frac{-fy}{z}) \tag{4.2}$$

where C is the image position of the camera center, in pixels, D is the size of image pixels in sensor coordinates, in $\frac{mm}{pixel}$, and s_x is a unitless scale factor to account for the mismatch between horizontal scan rates of the camera and the frame grabber. In some cases, such as robotic manipulation of the recognized objects Lloyd, Beis, Pai & Lowe (1997), a high degree of accuracy in pose determination is required, and a more complex model which includes radial distortion should be used. This can also be accommodated within the Tsai method (see (Tsai 1987)).

4.1.3 Synthetic Images and Hidden-Line Removal

Generating synthetic images of database objects is necessary both in the training stage, to generate training vectors for inclusion in the index, and in the recognition stage, for matching

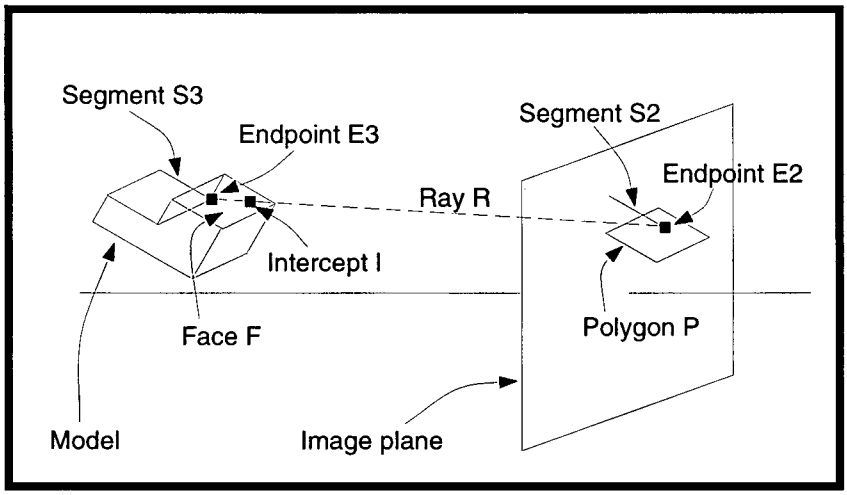


Figure 4.3: Hidden line removal.

the model with the image. Many recognition algorithms use wire-frame models for matching, but this increases the likelihood of false positive recognitions in cluttered images, by allowing matching of model edges that would be self-occluded. "Hidden-Line Removal" (HLR) creates more realistic model projections. Because we allow non-convex models, a simple test on the surface normal of the face is not sufficient to determine feature visibility in all cases, and a more complex HLR algorithm is required. (Back-pointing faces, however, can be quickly removed in this way.)

To check if a 3D segment is partially or completely obscured by a model face, both the segment and the face contour are projected into the image plane and checked for overlap (Figure 4.3). If either endpoint of the segment is in the interior of the projected polygon, the 3D model face is intersected with the ray R that emanates from the image point and passes through its generating 3D model point. The intersection point is labelled I in the figure. If I is closer to the camera than the 3D endpoint $E3$, then the face must occlude that portion of the segment that overlaps with it in the image, and the image segment $S2$ is clipped against the image polygon P using a packaged clipping algorithm (Glassner 1990).

This procedure is expensive because each edge is tested against each forward-pointing

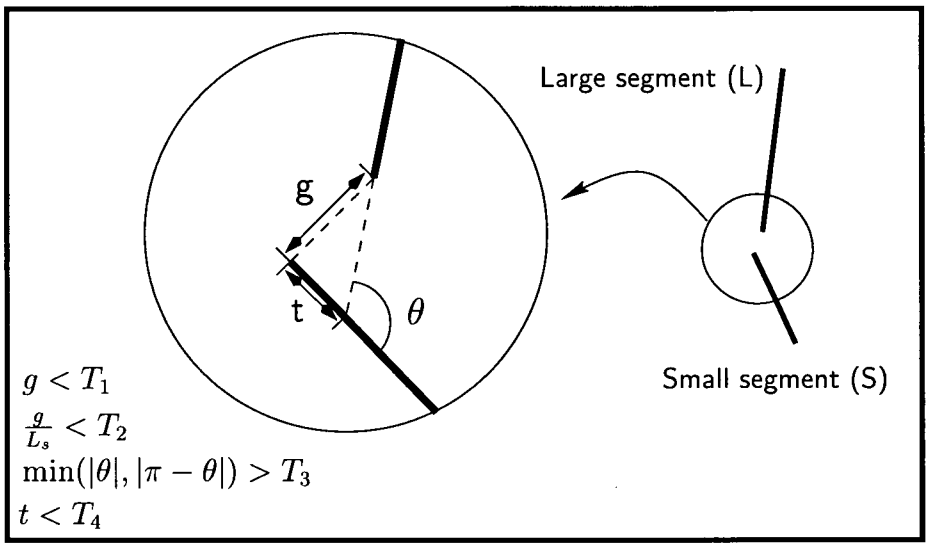


Figure 4.4: Definition of cotermination groupings. T_1 is the maximum absolute gap allowed between segment endpoints (in pixels); T_2 is the maximum relative gap, which depends on L_s, the length of the shorter of the two segments; T_3 ensures that the segments are not parallel (which is important since segmentation often breaks a long, straight segment into smaller pieces at positions which are unstable); and T_4 ensures that the junction is not a T-junction. Conservative values for these would be $T_1 = 10$ pixels; $T_2 = \frac{1}{2}$; $T_3 = 0$; $T_4 = 10$ pixels. We found that tighter thresholds provided faster recognition with only a small loss in grouping and indexing performance: $T_1 = 6$ pixels; $T_2 = \frac{1}{8}$; $T_3 = \frac{\pi}{100}$; $T_4 = 4$ pixels.

face to determine the visible segments. This is acceptable for the training stage, but in the verification stage the computation is performed at each iteration of least-squares pose determination during matching. The run-time process has been sped up with a lookup table created in the training stage. For each of the viewpoints used in training, a list of only those segments which were occluded is saved, along with the labels of the occluding faces. The lists are indexed by the viewpoint parameters, so that for any runtime model/pose hypothesis, the information from the closest training view can be quickly accessed. This means that only a small fraction of segments and faces need to be tested, and a large speedup is observed. Some mistakes can be made due to the gaps between the training views, but these are few and will seldom affect the final verification decision. Appendix A.1 gives the details of HLR complexity, and the average runtime speedup, which was determined empirically to be a factor of 6.9 faster.

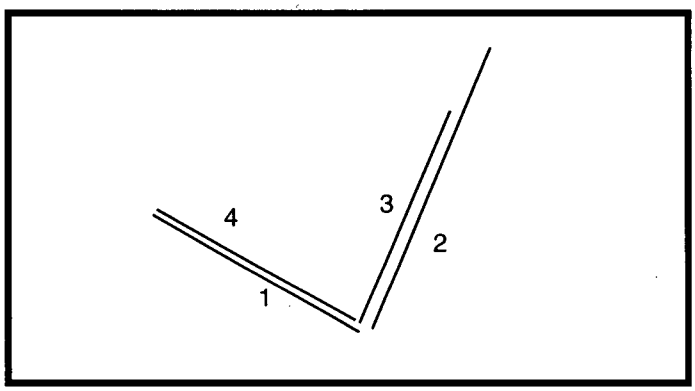


Figure 4.5: Corner detection ambiguity.

4.2 Grouping

4.2.1 Corner Detection

Coterminating segments, or corners, are the fundamental components for building segment chain groupings, which were the main grouping type used to test the indexing algorithm performance. To detect coterminations, the image is first divided into rectangular buckets, and segment endpoints are binned. For each segment s, a search for coterminating segments extends only to those buckets that lie within a certain distance d of the endpoints. To make the process scale-independent, d is taken to be some fraction of the length of s. There are several thresholds used to decide what is a corner (see Figure 4.4).

Ambiguities in corner detection arise when there are two parallel segments that could reasonably coterminate with another segment (see Figure 4.5). The current system does not try both possibilities, since several ambiguities could compound to significantly increase verification time. The choice of the "best" corner at this stage is important because an the error could be costly (a true positive index could be lost).

A segment with competing parallel edges is only grouped as a corner if it is rated as best for both of the participating segments. With reference to Figure 4.5, suppose segment 1 is processed first. Segment 3 is the closest cotermination, but since 3 has a better fit with 4,

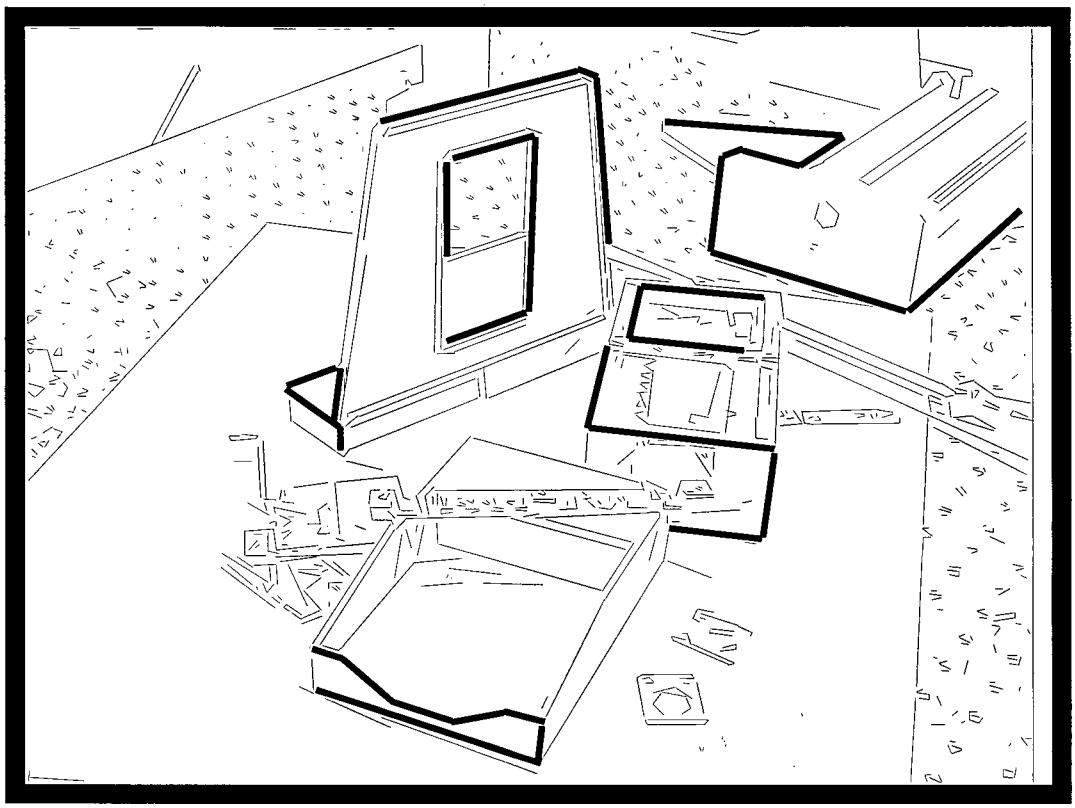


Figure 4.6: Examples of segment chain feature groupings (bold lines) of various cardinalities.

it is dropped from consideration. The remaining parallel segment (2) is then paired with 1, and later, 3 with 4. This procedure is not foolproof, but works in the vast majority of cases. More extensive reasoning, including disambiguation of larger sets of parallel edges, would be ideal. However, spending a great deal of time to make incremental improvements in grouping is probably less important than either (i) increasing grouping redundancy, so that a richer set of groupings is available for indexing, or (ii) postponing ambiguous match decisions until there are no other matches to add.

4.2.2 Segment Chains

Once corners are available, it is easy to generate longer chain groupings. Corners are stored in an array indexed by segment number. Each array entry is a list of segments that coterminate

with the indexed segment, typically contain two or fewer elements. The next link in a chain can be found by re-indexing the array using a segment label from a list entry. All unique chains of a given size are formed. A unique head and tail can be determined for each grouping using, for example, the smallest angle found at either end. For closed chains, all permutations are stored so that, if the cycle is broken in some image by noise or occlusion, indexing may still succeed. Examples of segment chain groupings are given in Figure 4.6. The computational complexity of grouping segment chains is given in Appendix A.2, and shown to be $\mathcal{O}(N^2)$, where N is the number of image segments.

The minimal number of points which overconstrain the pose (and thus provide discrimination in indexing) is four, corresponding to a minimal chain length of three segments. The canonical feature vector formed from a chain grouping of cardinality C_{ch} has length $2C_{ch} - 2$, and consists of $C_{ch} - 1$ angles between consecutive segments, and $C_{ch} - 1$ edge length ratios. Note that these features exhibit the partial invariance property mentioned in the previous chapter, and vary mainly with the two degrees of freedom associated with the view sphere. For greater noise resistance, it is possible to leave out the length ratios for the extremal segments, since endpoint locations are often eroded by the edge detection process. (The interior endpoints are accurately determined as the intersection point of the coterminating segments.) However, this lowers the dimensionality of index space by 2 for these groupings, and longer chains would be needed to achieve the same power of discrimination.

4.2.3 Parallel Segment Detection

Detection of parallel segment groupings proceeds as in the case of chains, by first grouping closest pairs of parallels, and then linking these into larger sets. A single "base" segment will typically be paired with at most one "external" parallel segment to either side. An exception to this occurs when there is no overlap between two external segments that fall to the same side of the base. Specifically, lack of overlap means it is possible to place a line perpendicular to the base segment that passes between the two external candidates, without intersecting either one.

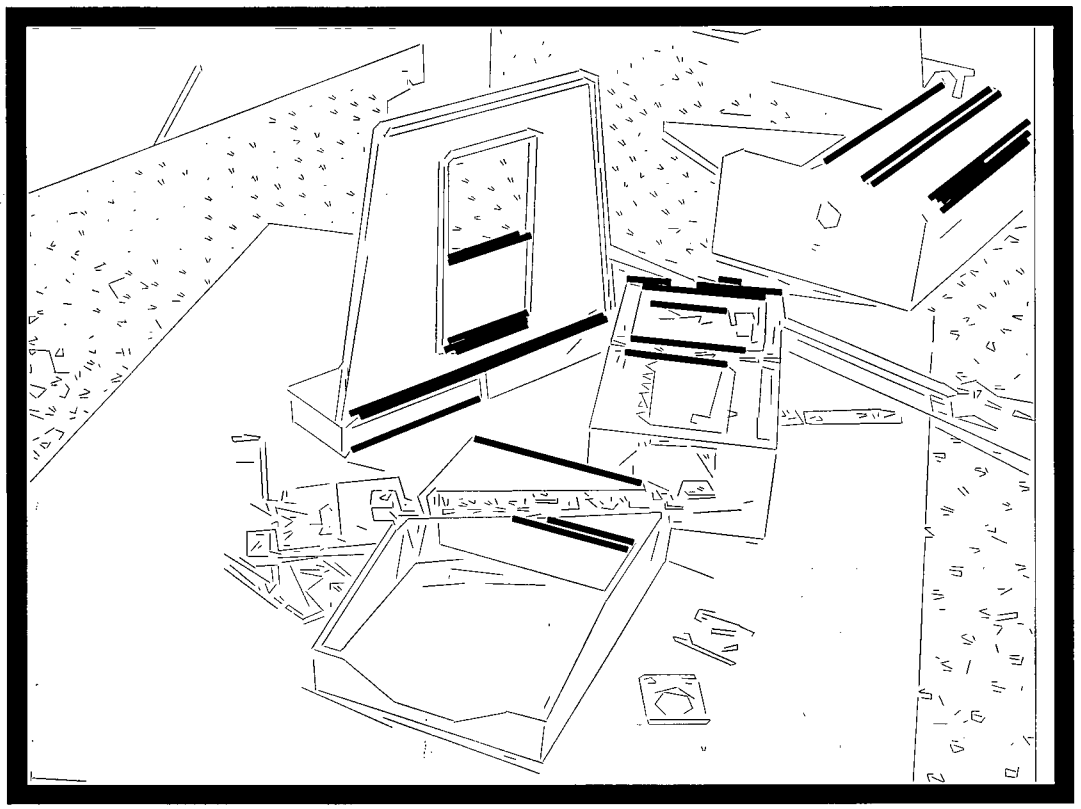


Figure 4.7: Examples of parallel feature groupings (bold lines) of various cardinalities.

Examples of parallel segment groupings are given in Figure 4.7.

The thresholds which define parallelism of segments are given in Figure 4.8. A grouping of cardinality C_p will give rise to a feature vector of size $3C_p - 3$, consisting of $C_p - 1$ edge length ratios for the segments, the same number of length ratios for the gaps g, and finally an equal number of offset angles ψ of the segment midpoints.

The set of all corners and all parallel pairs can be used to provide a rich set of groupings, by combining them in a variety of ways. The current set includes all cardinalities of homogeneous chain groupings and homogeneous parallel groupings.

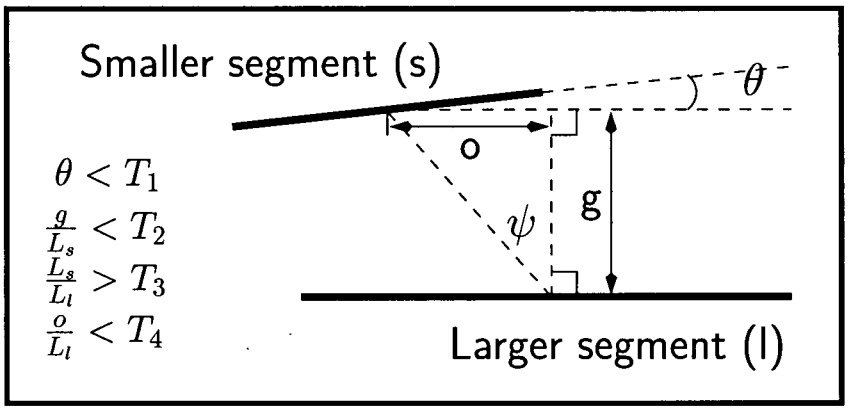
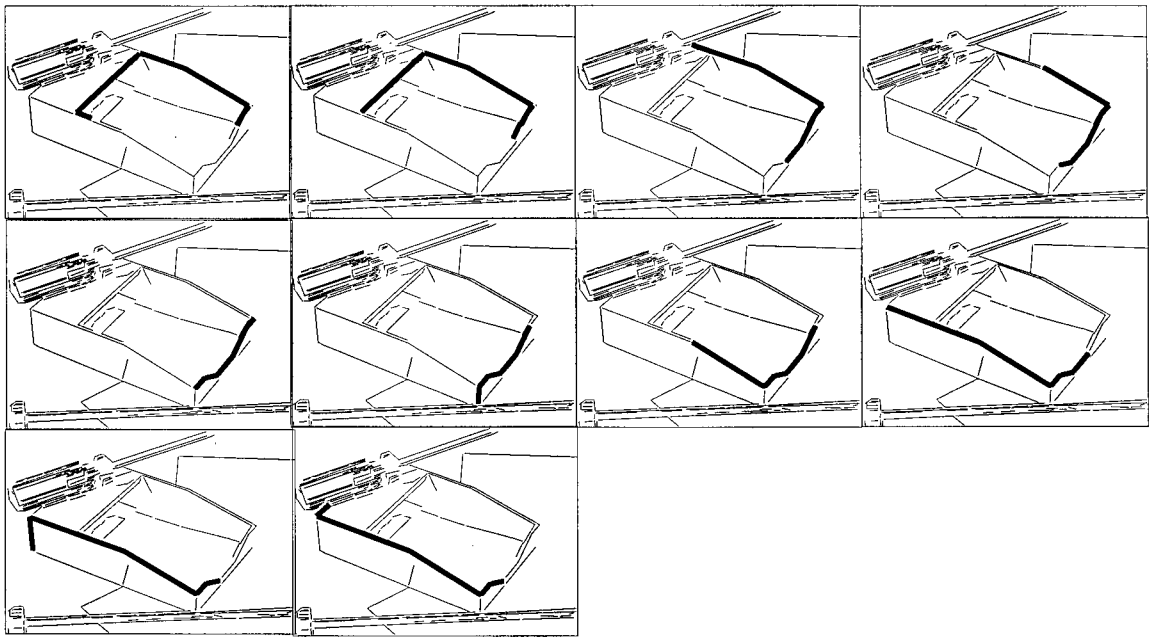


Figure 4.8: Definition of parallel segment groupings. Angular difference must be less than T_1; as above, T_2 is the maximum relative gap; T_3 ensures the segments are roughly the same scale; and T_4 ensures that the parallel segments overlap significantly. The values used in our system were: $T_1 = 5$ degrees; $T_2 = 1.0$; $T_3 = \frac{1}{4}$; and $T_4 = \frac{1}{2}$.

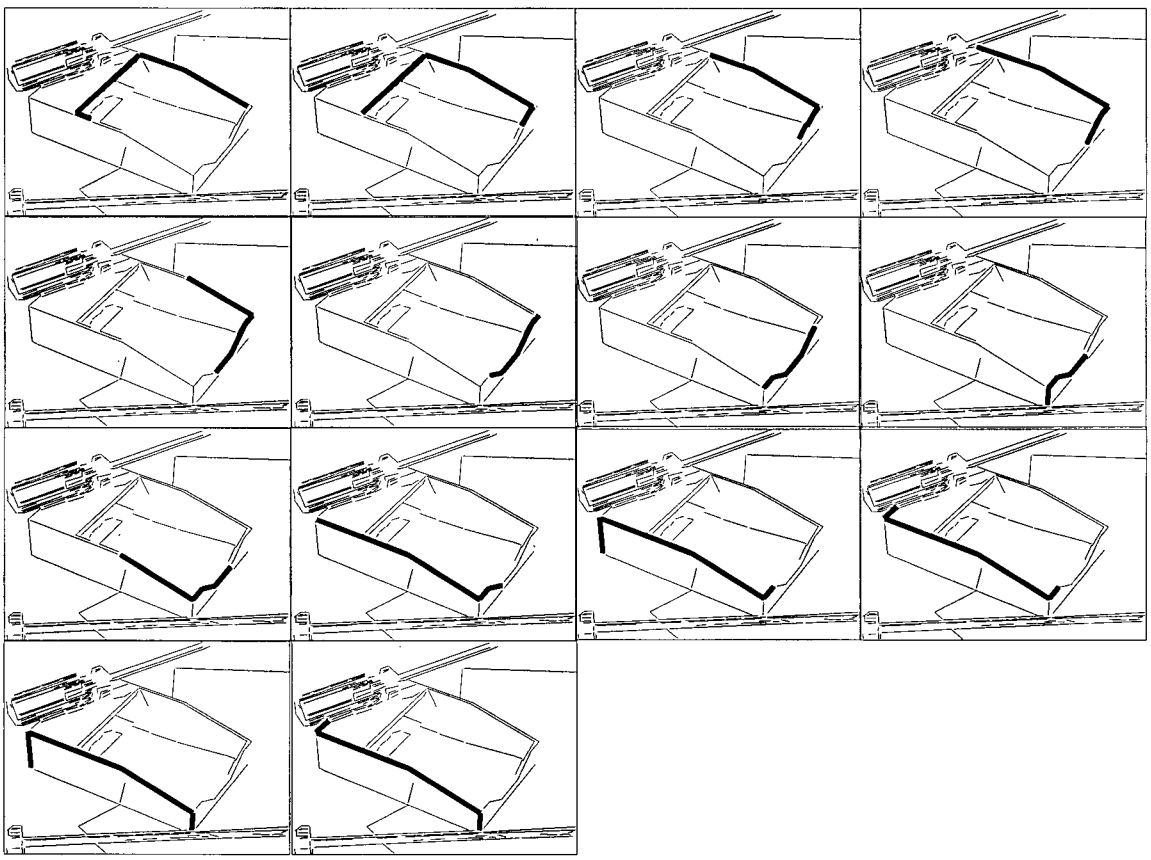
4.2.4 Failure Modes

Grouping is a critical step in the recognition process. Indexing will fail if there are no image groupings detected that correspond to a stored model grouping. Grouping may fail due to: large amounts of occlusion (unable to form large enough local feature sets); poor lighting (poor edge set from edge detection); or the object is in a degenerate position relative to the camera (few distinguishing features visible).

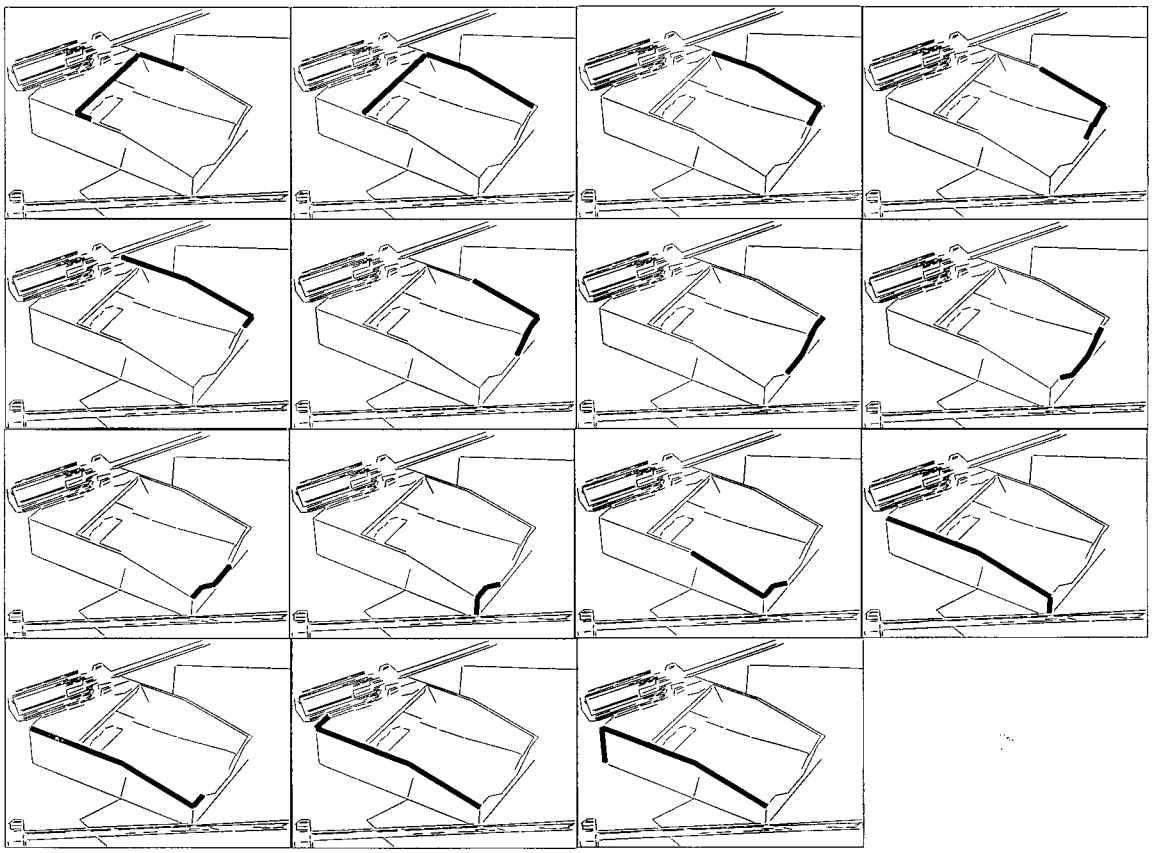
This is to be expected since, at some point any method, including human vision, breaks down in each of these cases. The current indexing system relies on redundancy of groupings and grouping types for each view of an object, and uses minimal groupings for indexing when conditions prevent detection of more salient feature sets (see Figure 4.8). An important aspect of a more robust system would be additional feature types, both in terms of different combinations of geometric features, as well as different modalities such as color and texture.



(a) Ten 5-chains.



(b) Fourteen 4-chains.



(c) Fifteen 3-chains.

Figure 4.8: Grouping trade-off: specificity vs. robustness. In this image, there are (a) 10 5-chains (b) 14 4-chains and (c) 15 3-chains that can be used to index the object. In difficult imaging conditions, it may be necessary to forego the specificity of larger feature vectors, and fall back on smaller groupings which are more robustly detectable.

4.3 Nearest Neighbour Lookup

The data structure which is used to store and retrieve feature vectors, the kd-tree, has been described in Chapter 3. In Chapter 5, we show that while the standard version of kd-tree search breaks down in high-dimensional index spaces, there exists an interesting *approximate* algorithm which makes indexing in these spaces practical. Higher-dimensional index spaces, which correspond to more complex feature vectors and more interesting shapes, are necessary for discrimination within a large model database.

In the approximate NN algorithm, the structure of the tree is unchanged, but the order

of the backtracking search for nearest neighbours is altered so that the leaf nodes closest to the query point are checked first. Setting a limit on the number of leaf nodes that may be visited means that, in some cases, the exact nearest neighbour will not be found. However, we show that this is unlikely for spaces with up to 20 or 25 dimensions, even when the search checks only a small number of leaves. When the best neighbour is not found, the method does not fail completely, but finds a very close neighbour instead. This "Best Bin First" (BBF) search is used in the experiments later in the chapter. Hash table lookup, an alternative preferred by many competing indexing approaches, is shown to be infeasible in high dimensional spaces.

4.4 Probability Estimation and Ranking

Probability estimates were computed by three of the four methods outlined in Chapter 3: k-nearest neighbours, WkNN, and Radial Basis Functions. Parzen windows were not used due to their inefficiency, since there can be a very large number of points stored in the index. Each method has certain parameters to set, and some implementational choices.

4.4.1 Kernel Function (WkNN and RBFs)

The main property that a kernel should have for this application is limited support, since the probability density function for any grouping should be zero far from its sample set. As well, the window width should be variable so that the smoothness of the interpolation can be adjusted. A final useful quality for the kernel is differentiability, since VSM or other descent-type algorithms may be applied to optimize the relative dimensionality weights for the system.

In function approximation, Franke (1982) and Broomhead & Lowe (1988) find the "multiquadric" function $f(x) = \sqrt{c^2 + x^2}$ to perform well, although this kernel has unlimited support. The limited support "inverse multiquadric" $f(x) = (c^2 + x^2)^{-\frac{1}{2}}$ also provides good results (Franke 1982). In this thesis the interpolating kernel is chosen to have a Gaussian form, which satisfies the criteria mentioned above. (Although in theory Gaussians have infinite support,

in practice support is limited by the rapid fall-off.) The ability of a given kernel to generate accurate probability estimates is highly dependent on the form of the underlying distributions, and in our case these are currently unknown. Differences between kernels are generally not dramatic in any case, and if necessary this choice can be optimized later.

4.4.2 Dimensionality Weighting

Each of the proposed estimation techniques depends on the distance metric in the index space. Scaling one dimension with respect to the others changes the notion of similarity and distorts probability estimates. Dimensionality weights should be set so that the predictive value of a point for classification is the same at all locations equidistant from the point.

The difficulty arises when dimensions refer to disparate quantities. For segment chain groupings, there are two distinct feature types, which are measured in different units: angles (in radians) and edge length ratios (unitless). However, for purely geometric features, and possibly for others as well, we believe that it is possible to set reasonable values by hand. These values are chosen using intuitive notions of similarity, knowledge of image noise levels, and variation of feature values between nearby training views, so that, for example, a 5 degree difference in angle is considered a small shape difference, but 20 degrees constitutes a large change of shape.

To test this, a modified version of the VSM learning technique (see Section 3.6.3) was used, in which the kernel size was fixed. In standard VSM, the kernel width depends on the recovered set of neighbours. In sparse regions of the space, the kernel expands to interpolate the neighbours, supplying the best classification decision possible given the small amount of data. However, this means the notion of similarity varies from query to query. Because we compute density estimates for each image feature vector, and later compare results *between queries* to determine hypothesis rankings, it is important that the similarity measure remains constant.

VSM was used to test the by-hand settings of weights in a $4D$ index space, which was generated from 4-segment chain groupings using the three interior angles and the edge length ratio of the two interior segments. VSM confirmed that all angle weightings should be set to

roughly the same value (as per our original guess), although the relative weighting for the edge length ratio feature was determined to be about half as large as originally estimated. The performance improvement, which in VSM is measured by classification error, was negligible however, so it appears that the original values were not much worse than the optimized values.

4.4.3 Kernel Width (WkNN and RBFs)

In the previous section the *relative* weights of the dimensions were considered. In the WkNN and RBF approaches, there is an extra parameter, the window width (Gaussian σ), that sets the *absolute* scale of the space, and determines the smoothness of the interpolation. A larger value for σ means that interpolation may occur between more widely spaced data samples, but that in densely-sampled regions, higher-frequency information is lost.

In addition to the dimensionality weightings, VSM can also be used to optimize the kernel width. Performance was found to be very insensitive to kernel width, which may indicate that the optimal value varies from place to place within the space. There is currently no effective way of capturing this variation.

4.4.4 Value for k (kNN and WkNN)

This is similar to the choice of kernel width, since the value of k determines an effective volume over which to average samples. The difference is that the volume is data dependent. A larger value for k means that noise is reduced and estimates are smoother, but that higher frequencies may be lost. In our experiments, k was set to 10 (unless otherwise specified), and this provided satisfactory performance. To our knowledge, there are no theoretical results on how to set the absolute value of k, and optimization is an empirical procedure. (Duda & Hart (1973) show how the relative value of k should grow with the amount of data.) VSM might be used to determine a value for k, by matching the volume implied by the optimal kernel width (above) to the volume implied by a set of k neighbours (distance from query point to k^{th} neighbour), averaged over a large set of samples in index space.

4.4.5 RBF Implementation

Implementation of Radial Basis Functions for probability estimation requires inversion of an $N \times N$ matrix, where N is the number of stored points. This matrix is sparse, so it is likely that with some manipulation, the inversion time can be kept reasonable. However, because good results can be achieved with the simpler estimation methods (kNN and WkNN), a more basic RBF approach was implemented that is closer to the original, function approximation method.

For each $3D$ grouping g, an RBF is constructed from the sample set of data points:

$$F_g(\vec{\mathbf{x}}) = \sum_{i=1}^{N_g} c_{gi} G(\|\vec{\mathbf{x}} - \vec{\mathbf{x_i}}\|). \tag{4.3}$$

As in the WkNN method, for each index query a set of k neighbours is recovered, but rather than weighting with Parzen kernels, weightings are determined by the RBFs:

$$p(g|\vec{\mathbf{x}}) = \frac{\sum_{i=1}^{k_g} F_g(\vec{\mathbf{x_i}})}{\sum_{i=1}^{k} F_{g_i}(\vec{\mathbf{x_i}})}. \tag{4.4}$$

Note that each RBF includes information from all samples of a grouping, so the number of data points involved in the probability estimates can be much greater than k.

The computation of RBF coefficients c_{gi} involves the inversion of the matrix with elements $G_{ij} = G(\|\vec{\mathbf{x_i}} - \vec{\mathbf{x_j}}\|)$ for $i, j = 1...N_g$. This matrix is singular if two of the data points are identical, and if two points are merely close together, the inversion becomes unstable. This can be avoided by adding to the diagonal elements of the matrix (which have value 1.0) a stabilization parameter $0.01 \leq \lambda \leq 0.1$.

4.5 Verification

Match hypotheses recovered from the index indicate *local* shape matches between object and image. Each of these must be verified to ensure that the hypothesis does not correspond to a coincidental alignment of image features, or a local shape on some other object. This is a 2-stage procedure, that iterates between an alignment stage, which determines a pose estimate, and a match extension stage, which adds further correspondences into the current set.

4.5.1 Alignment

Alignment is achieved using Lowe's (1987a, 1991) method which, given a rough initial pose estimate for a set of correspondences, iteratively solves for a best least-squares fit of the model to the image, using a version of Newton's method. One advantage of non-invariant indexing over invariant indexing and other methods of hypothesis generation is that a pose estimate is available and stored with the training data. In our indexing approach, several neighbours may contribute to a given match hypothesis, but it is sufficient to use the pose from the single recovered model vector that is closest to the image query vector.

Lowe's method minimizes an error function

$$E = \sum_{s=1}^{S} \sum_{e=1}^{2} d_{\perp}(M_{se}, I_s) \tag{4.5}$$

where s subscripts the corresponding model/image segments, of which there are S; e denotes endpoint within a segment; and $d_{\perp}(M_{se}, I_s)$ represents the perpendicular distance of projected model endpoint M_{se} from image segment I_s. In effect, segments are being matched rather than points: endpoint offsets are not penalized so long as the segments are collinear. This is useful because endpoint locations are quite uncertain due to edge detection artifacts, and image segments may be broken or eroded. (Unfortunately, it also allows for a perfect match between segments that have no mutual overlap.)

Lowe (1987a) introduces a parametrization which allows for the simple and efficient application of Newton's method to the problem. The initial pose estimate mentioned above is required because Newton's method solves at each stage a linearized version of a non-linear problem, which is only locally valid. Lowe (1991) extends the approach, using the Levenberg-Marquardt method to stabilize the solution. This involves a parameter λ that automatically switches between gradient descent (guaranteed but slow convergence) and Newton's method (faster, but with no guarantee of convergence) by monitoring the progress of the residual at each iteration. The stabilization ensures convergence for a wide range of values of the initial parameter guesses, and it was noted in our experiments that, for valid hypotheses, the method

always converged to a reasonable solution.

4.5.2 Match Extension

Once the best fit to the current set of matching features has been computed, the pose estimate is used to project the model back into the image and search for further matches. Ideally, for a valid hypothesis, a single alignment iteration would provide an accurate pose estimate. However, there are at least two reasons why this often fails, and why it is therefore necessary to iterate between alignment and match extension:

(a) There are some "degenerate" configurations of features that can cause problems. For example, if the 3D feature set is planar, then substantial out-of-image-plane rotations, coupled with a scaling or z-translation, can leave the error E almost unchanged[1]. This is one negative aspect of matching with lines rather than points. Coupling this property with image noise can lead to a poor pose determination.

(b) Small image groupings, both in cardinality and in spatial extent, may give poor results in the presence of noise.

Figure 4.9 shows an example of a skewed alignment based on an initial set of four corresponding segments. Note that model edges further from the correspondence set are misaligned to a greater extent than closer ones (4.9(b)). The problem is that if even one of the distant segments becomes matched the verification may fail, because least-squares matching is not robust to outliers.

To handle these situations, on the first few iterations new matches are included only if they coterminate with a segment in the current match set. This is a crude but effective way of ensuring that only proximal segments are added in the early stages of matching. In general, if the hypothesis is valid, the pose rapidly converges and the restriction can be relaxed. Final

[1]This is different from the *2-fold degeneracy* of any planar feature set, which looks identical when slanted towards or away from the image plane. The degeneracy is exact for orthographic and weak perspective projection, and for full perspective is typically within match tolerances. The only solution is to try both alternatives.

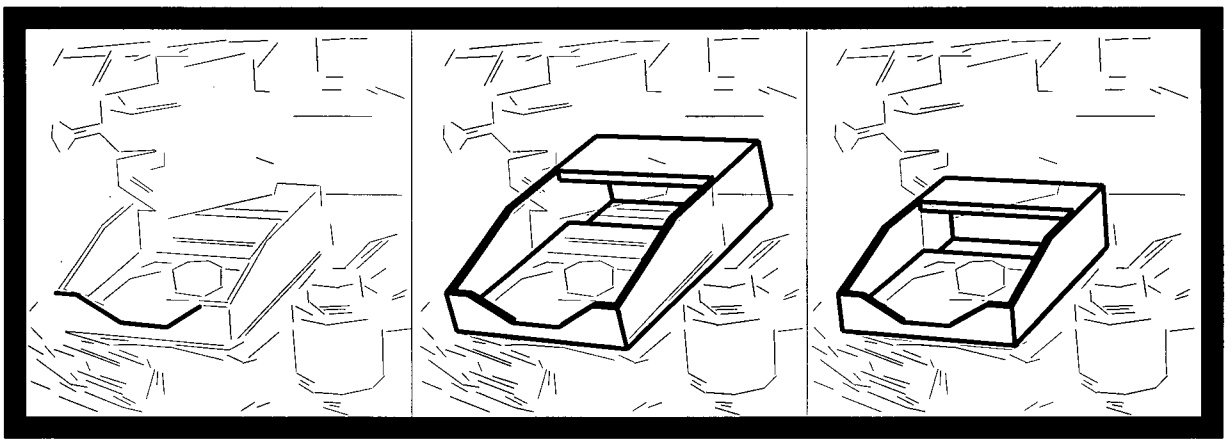


Figure 4.9: Example of match extension process. (a) Initial correspondence hypothesis. (b) Initial least-squares match. (c) Final match using match extension procedure.

match accuracy (4.9(c)) can be very accurate because of the large amount of data used in the computation.

4.6 Performance Experiments

There are two questions to ask regarding the performance of an indexing and recognition system: (i) How often is the object found? and (ii) How long does it take? Each of these questions can be further subdivided into two parts: (a) The performance of the indexing mechanism, and (b) The performance of the rest of the recognition system. In this section (and in the "experiments" sections of Chapters 5 and 6), we investigate the degree to which each component can be said to work, and provide estimates of the time required for each.

4.6.1 Synthetic Data

Synthetic images of database models can be used as test images, to isolate the indexing mechanism from the other aspects of recognition. Since ground truth is available, it is possible to determine when indexing has succeeded even if subsequent verification were to fail. The database of 10 object models is shown in Figure 4.10. The variables to be investigated are (i)

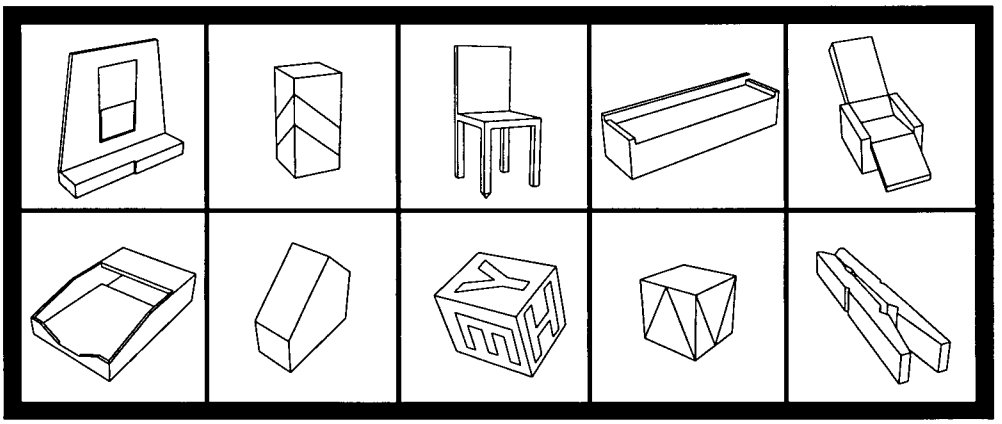


Figure 4.10: Model database.

the number of database objects; (ii) the number of views of each object; (iii) the dimension of the index space; (iv) the noise level in the images; and (v) the method of probability estimation.

All experiments in this section followed the same basic procedure. First, a set of training images were generated, using either a tessellation of the view sphere (240 views), or random views sampled uniformly over the view sphere. Grouping methods were applied to each image and the resulting training vectors stored in the (kd-tree) index. Each test image was of a single database object, chosen at random and viewed from a random viewpoint. Because some objects have two-fold symmetry, test views were restricted to a hemisphere: otherwise, a valid match to a symmetric counterpart grouping would be rejected because ground truth segment labels didn't match.

Gaussian noise was added to all test images. Each segment endpoint was perturbed in both its x and y image coordinate, at a level controlled by the standard deviation σ of the Gaussian, measured in image pixels. σ was set to 1.5 pixels for all experiments other than the experiment testing noise performance. This level was chosen subjectively as the typical level found in real images.

The search method used was the BBF approximate kd-tree search (to be detailed in Chapter 5). The number of nearest neighbours to retrieve, k, was set to 10, and the maximum

		1 model	5 models	10 models
# POINTS	PLL3	1,900	9,600	20,000
	CHAIN3	8,700	60,400	133,000
	CHAIN4	11,400	93,400	211,000
	CHAIN5	13,500	134,000	303,000
	CHAIN6	15,300	190,000	438,000
# GROUPINGS	PLL3	200	1,100	2,000
	CHAIN3	300	2,800	5,500
	CHAIN4	500	6,700	13,600
	CHAIN5	800	13,400	26,900
	CHAIN6	1,100	24,300	49,700

Table 4.1: Index size (number of stored points), and the number of underlying 3D model groupings that give rise to those points, for various database sizes. Training was with 240 views per model. PLL3 are 3-segment parallel groupings with a 6D index space; CHAINn groupings are n-segment chain groupings with $2(n-1)$-dimensional index spaces.

number of leaf nodes examined per query was 100. For a single run, each data point was generated by averaging over 10 test images per database model. Each experiment was run 10 times with different sets of random test images.

Two performance measures were used.

- **Percent found**: The percentage of time that the complete set of hypotheses contained at least one correct match. A hypothesis was considered valid as long as the correct model grouping – image grouping match was indicated, even if subsequent verification might have failed.

- **Index ranking**: For those images in which the object was found, the rank of the first correct hypothesis.

While the scenario of synthetic images with no clutter is somewhat artificial, it provides insights into the following questions: How does system scale with index space dimension? With the number of database objects? Does the system get confused by the large number of stored points? (Table 4.1 shows, for each experimental situation, the size of the index.) Does the ranking work to reduce the number of verifications required?

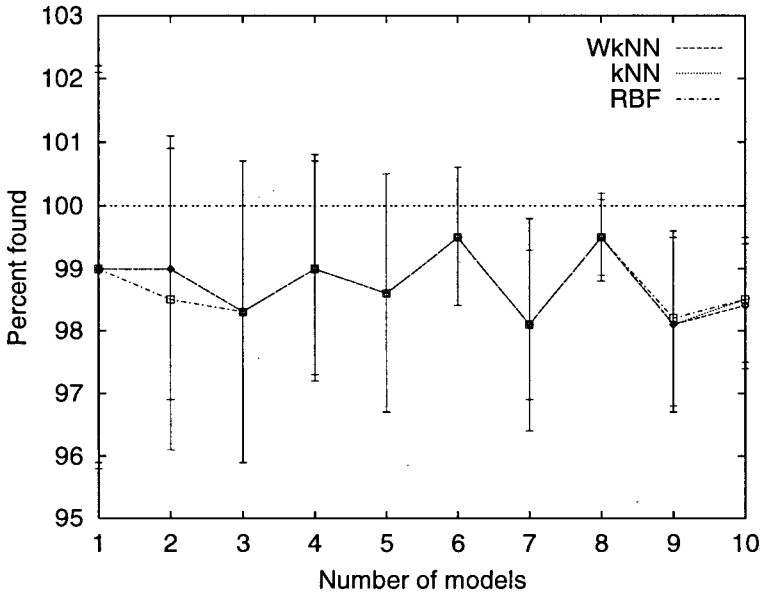


Figure 4.11: Performance vs. Number of models: Percent found.

Number of Models

In this experiment: tessellated training was used (240 views over sphere); the grouping type was the 4-segment chain, which has a 6D index space; the number of models varied between 1 and 10. As well, the three modes of probability estimation were compared.

The three methods are virtually indistinguishable with respect to the percent found (Figure 4.11). The small number of objects not found correspond to "degenerate" views, in which there are few groupings visible, none of which is salient with respect to the model database. An example of this would be an end-on view of an object, in which only one face is visible.

kNN and WkNN do slightly better than RBFs in ranking the hypotheses (Figure 4.12), but all three indicate that very few hypotheses need to be verified before the object is found. Recall that without indexing, each image grouping would have to be compared to all groupings from all models, which is a difference of several orders of magnitude.

It is somewhat surprising that kNN and WkNN give such similar results. From a closer inspection of the data, it was noted that for some queries (low density region of index space),

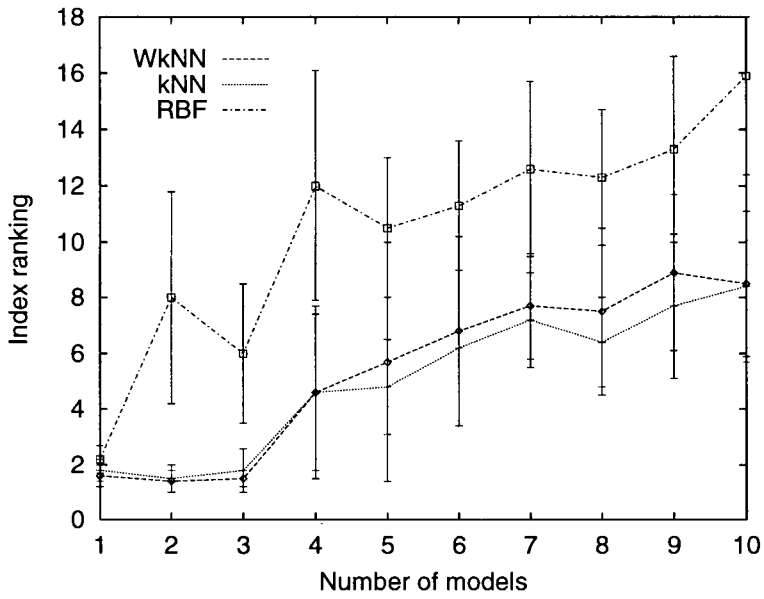


Figure 4.12: Performance vs. Number of models: Index ranking.

kNN provided a higher ranking than WkNN to a correct match, when some of the distant neighbours in the recovered NN set agreed and were valid matches. In those cases, the roll-off of the Parzen kernel had a negative effect. However, the converse was also observed (also in low density regions), that kNN could provide a lower ranking to a correct match if a small number of close neighbours indicated the true positive, but several of the far neighbours agreed and indicated a false positive. Then the Parzen kernel properly gave less weighting to the distant neighbours.

This demonstrates that a global setting for the window size is not ideal, since the rate of change of the underlying distributions varies from place to place throughout the index space. In the former case, a larger Parzen window would have been appropriate; in the latter, the smaller one was correct. While more difficult to observe, a similar effect undoubtedly occurs for the global settings of the dimensionality weightings. Unfortunately, there are currently no standard methods to generate local settings for these parameters.

The slow increase in ranking with number of models is important to demonstrate the

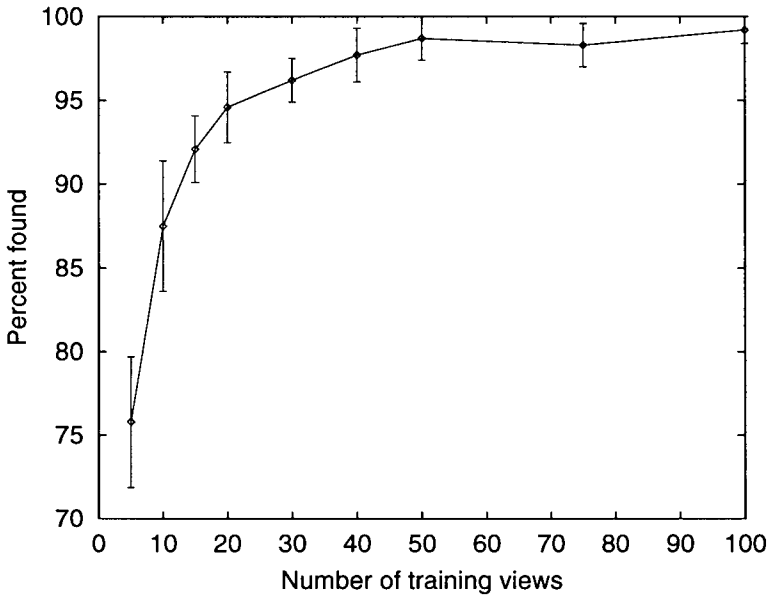


Figure 4.13: Performance vs. Number of views: Percent found.

scaling properties of the system. It indicates that, as the number of distractors in the index grows, the probabilistic rankings keep the number of verifications low. For example, with 10 models there are over 200,000 stored points (4-segment chain groupings), but the selectivity of the index is such that on average less than 10 hypotheses need to be investigated before a successful match is found.

Number of Views

In this experiment: training views were randomly sampled over a hemisphere (rather than the full view sphere, due to the problem with two-fold symmetric objects mentioned earlier); the grouping type was the 4-segment chain, which has a 6D index space; the WkNN mode of probability estimation was used; and the number of test images was 100.

Interestingly, the performance with respect to percent found is very good with even a fairly small number of training views, correctly indexing over 90% when only 15 views per hemisphere were used. This indicates that, over the full view sphere, perhaps only 30 to 50

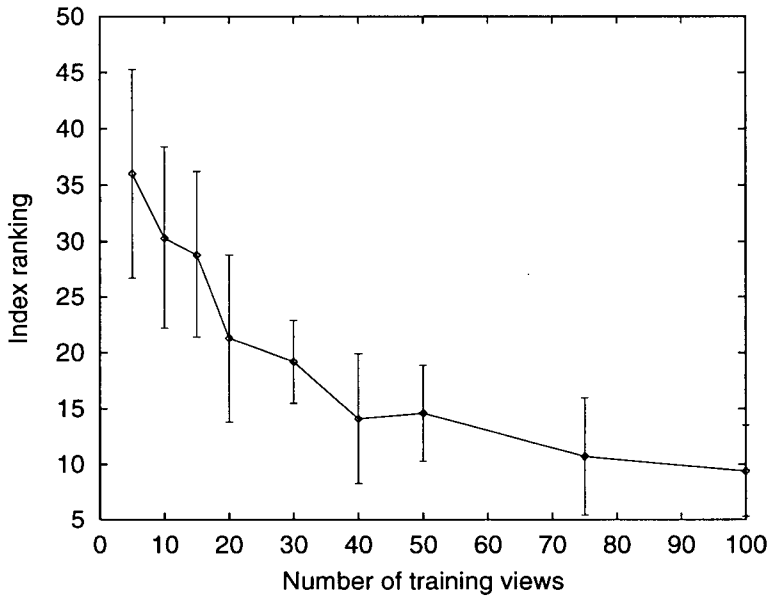


Figure 4.14: Performance vs. Number of views: Index ranking.

prototypes are necessary to accurately represent the probability distributions of most groupings, especially if their positions and weights are determined by a weighted clustering method such as WVQ (Chapter 6).

The improvement in rankings with number of views means that the probability estimates are working as expected. For most other methods, more distractors would mean *worse* performance because more hypotheses would be generated.

Dimension of Index Space

In this experiment: tessellated training was used (240 views over sphere); the WkNN mode of probability estimation was used; and the number of test images was 100. The grouping type was the n-segment chain, which has a corresponding $(2n-2)$D index space;

Again, the overall performance is strong. The slow decrease in percent found with dimension is possibly due to it being more likely in higher-dimensional spaces that a small change in viewpoint leads to a large change in at least one of the features. The interpolation

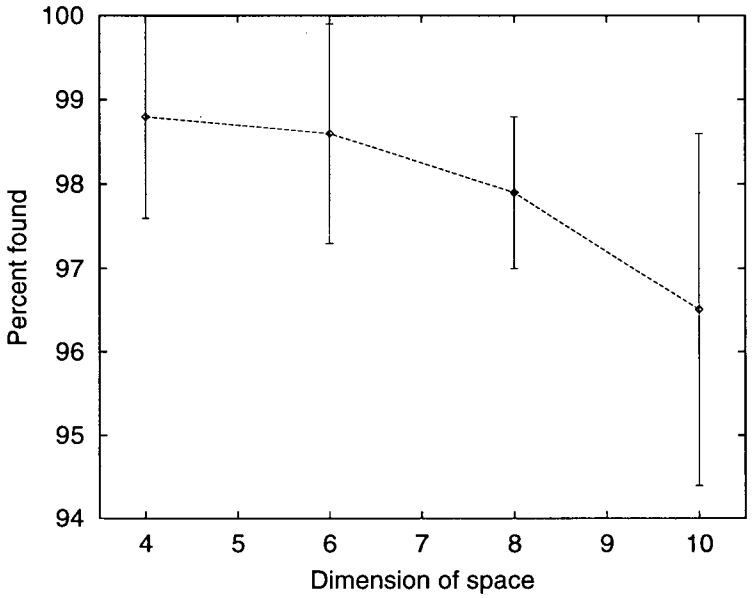


Figure 4.15: Performance vs. Dimension of index space: Percent found.

may therefore have failed for some of the more widely spaced (in index space) views. The remedy would be to increase the number of training views, or possibly the size of the interpolating kernel. It is also true that the approximate NN search is less effective as dimensionality increases, missing a slightly larger fraction of the closest neighbours, and this might contribute to the effect as well.

The decrease in ranking is due to the increased specificity with dimension, although the initial specificity is already quite high, so the improvement is slight. This effect would be more noticeable with a larger model database.

Image Noise Level

In this experiment: tessellated training was used (240 views over sphere); the grouping type was the 4-segment chain, which has a 6D index space; the WkNN mode of probability estimation was used; and the number of test images was 100. The standard deviation σ of the Gaussian noise for each endpoint, in both image dimensions simultaneously, was varied between 0 and

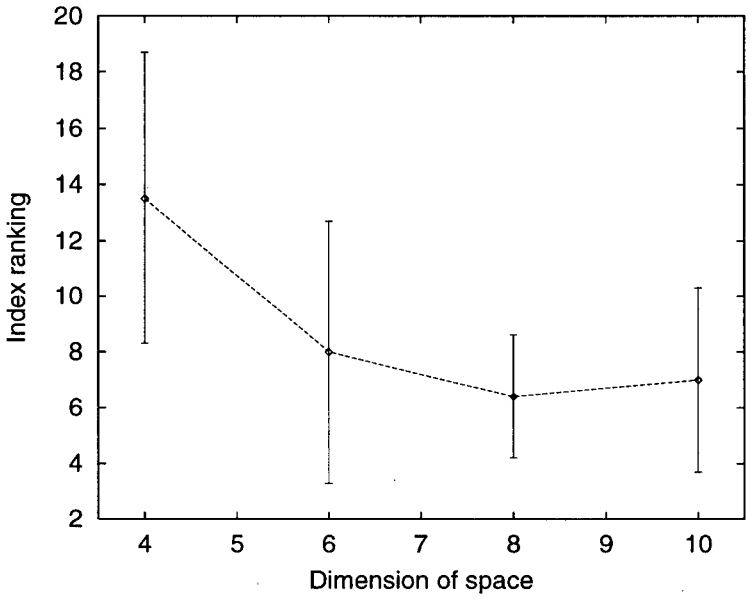


Figure 4.16: Performance vs. Dimension of index space: Index ranking.

4 pixels. Relative feature positions and orientations in the $\sigma = 4$ images were observed to be significantly more distorted than features in typical real images.

The slow decrease in percent found and the slow increase in ranking both indicate the graceful degradation in indexing function response with increasing noise, due to the availability of "smooth" probability estimates for the indices. Moving a point about in feature space by a small amount distorts probability estimates and rankings somewhat, but apparently this effect is not dramatic. Of course, these results are dependent on the size of the object in the image, and would be quite different for smaller objects.

As a final note, the large error bars on plots in this section most likely indicate the intrinsic variability of the performance measures from run to run, which is due to the variation in the image sets, rather than an uncertainty in the *average* values which appear, from the smooth nature of the curves, to be reasonably accurate.

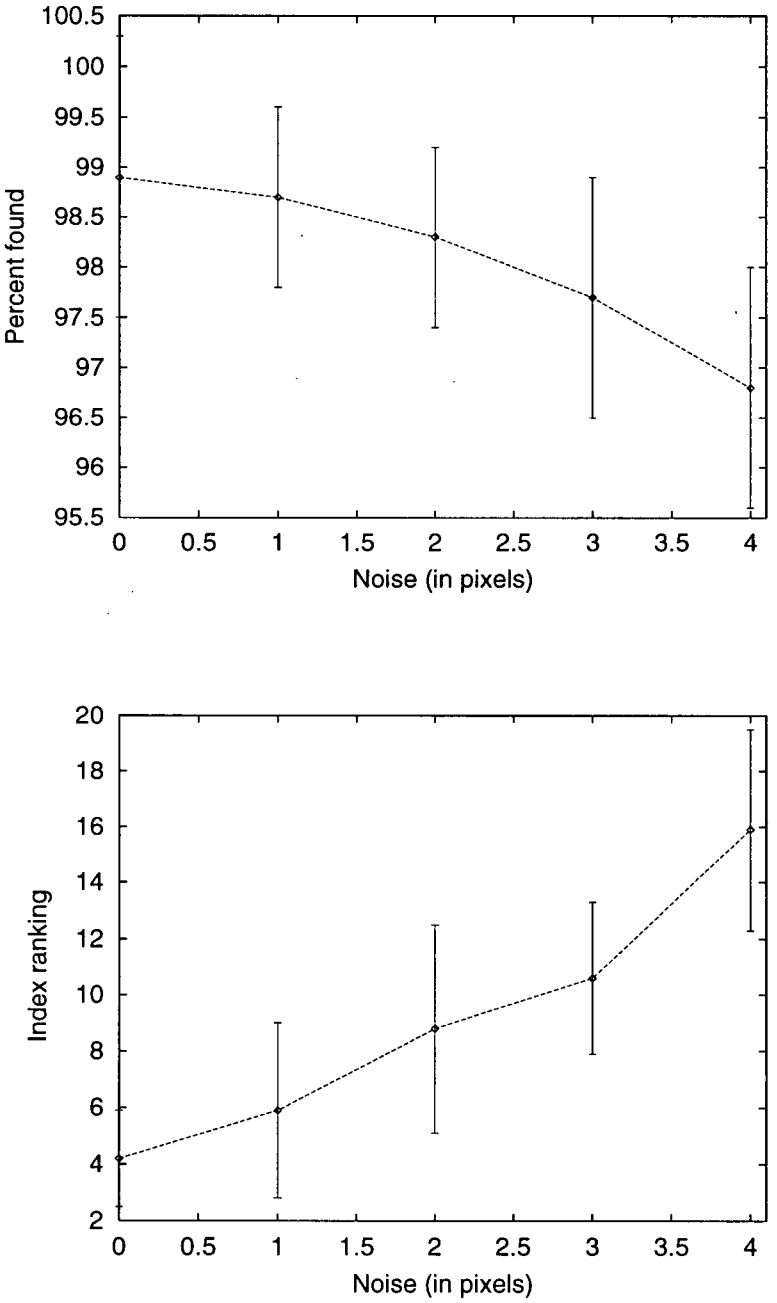


Figure 4.17: Performance vs. Image noise level.

4.6.2 Timing

The synthetic image experiments give relative timing information, that is, they indicate how the rankings reduce the number of verifications required. Here we present some absolute timing experiments using real image data. (For an analysis of the theoretical complexity of indexing with our method, see Appendix A.3.)

The recognition process can be subdivided into three stages: grouping, indexing, and verification. Some definitions are required:

(a) Let G be the average time to perform grouping, starting with a set of image segments.

(b) Let I be the average time to perform indexing with a single image feature vector. This is the time to recover a set of nearest neighbours from the index, and to form a probability estimate based on those neighbours.

(c) Let V be the average time to verify a single match hypothesis against the image.

(d) Let N_i be the number of image groupings.

(e) Let N_m be the number of underlying, 3D model groupings.

(f) Let k be the number of neighbours to recover from the index.

We can look at the time for recognition in three cases: (i) with indexing, worst case; (ii) with indexing, average case; and (iii) without indexing. The total time required will be the sum of the times for the three major components, grouping, indexing, and verification. The first component, grouping time, is the same in all three cases, simply G. For the second component, indexing, in cases (i) and (ii), the number of queries to the index will be equal to the number of image groupings N_i, so the total indexing time will be $N_i \cdot I$. In case (iii), indexing time is zero.

For the third component, verification, in our approach to indexing, the worst case occurs if each recovered neighbour leads to a different hypothesis, and all hypotheses must be investi-

gated. This leads to $N_i \cdot k$ verifications. The average case is only meaningful if the task we are performing implies that verification may terminate when an object is found. The number of verifications will then depend on the parameter set, as well as the particular image. In case an object is present in an image, this number (say A) will be much smaller than the worst case. Examples of typical values for A under various imaging conditions are given in the previous section, for synthetic images, and the subsequent section, for real images (the "index ranking" values). With no indexing method, all combinations of groupings must be tested, a total of $N_i \cdot N_m$ verifications.

The expressions for total time are therefore:

$$\textbf{Worst case:} \quad (G + N_i \cdot I + N_i \cdot k \cdot V) \tag{4.6}$$

$$\textbf{Average case:} \quad (G + N_i \cdot I + A \cdot V) \tag{4.7}$$

$$\textbf{No indexing:} \quad (G + N_i \cdot N_m \cdot V) \tag{4.8}$$

The average observed time for grouping 4-segment chains in 15 different images, with an average of 1100 segments per image, was $G = 120$ msec. On a variety of images, the average query lookup time was $I = 2.5$ msec, for a database with over 200,000 stored points. The average verification time per hypothesis was 25 msec. Typical values for the remaining variables might be: $N_i = 1000$, $N_m = 10,000$ (for a 10-model database), $k = 10$, and $A = 50$ (the latter being the worst case from all synthetic and real image experiments).

These values lead to the following absolute time estimates:

$$\textbf{Worst case:} \quad .12 + 1000 \cdot .0025 + 1000 \cdot 10 \cdot .025 = 250sec \tag{4.9}$$

$$\textbf{Average case:} \quad .12 + 1000 \cdot .0025 + 50 \cdot .025 = 3.9sec \tag{4.10}$$

$$\textbf{No indexing:} \quad .12 + 1000 \cdot 10,000 \cdot .025 = 250,000sec = 70hours \tag{4.11}$$

The numbers for the "no indexing" case are worst-case numbers, but the average case will only be reduced by a small factor, equal to the number of valid grouping matches. All timing experiments are for ANSI C code running on a SUN UltraSPARC-2 workstation.

4.6.3 Real Images

It is virtually impossible to do a thorough quantitative analysis of indexing and recognition algorithms, since it is problematic to define the overall difficulty of a given image for recognition. The variables of lighting, occlusion, noise, camera position, etc., all conspire to create an overwhelming world of possibilities. It is even difficult to quantitatively compare the most similar of the current indexing methods, as researchers use different types of image data and different model representations.

For these reasons, we organized the real image experiments in a partly qualitative, partly quantitative manner. We loosely define three qualitative regimes, for which example images are presented, and within which quantitative analyses are undertaken. All of these regimes use actual camera images, as opposed to rendered images. The "easy" image regime consists of images with very little clutter, no occlusion, and large objects (Figure 4.18, top). The "moderate" regime has images with significant clutter, moderate occlusion, and a variety of object sizes (Figure 4.18, bottom). Finally, "difficult" images are those containing significant clutter, moderate to significant occlusion, smaller objects, and a large number of features (Figure 4.19).

The training stage was the same for all three regimes. The optimal grouping for trading off specificity and robustness was found to be the 4-segment chain, followed closely by the 3-segment chain, and these are used in the experiments below. 3-segment parallel groupings were found to be useful for augmenting the chain groupings, but were not in themselves effective for indexing. The 240-view tessellated training method was used for all 10 models. This provided a total of 211,000 vectors from 13,600 groupings when using 4-segment chains; 133,000 vectors from 5500 groupings when using 3-segment chains; and 20,000 vectors from 2000 groupings when using 3-segment parallel groupings.

A simple verification criterion was used, in which a match was accepted if greater than 50% of the lengths of visible model segments were matched. We consider false negatives to be the critical issue for an indexing method, while false positives fall more within the realm of

	Easy	Moderate	Difficult
% found — 4-CHAINS	95	76	31
% found — ALL TYPES	99.5	96	50
# Hypotheses/image	165	400	1000
Average rank	2.7	10.2	158
Median rank	1	2	44

Table 4.2: Performance of indexing with real image data. "4-CHAINS" refers to indexing using only 4-segment chains; "ALL TYPES" refers to the use of both 3- and 4-segment chains, as well as 3-segment parallel groupings.

verification. Therefore, in discussions of experiments below, we concentrate on the ability of the index to avoid false negatives. In fact, the system typically generates few false positives, however one area of future work will necessarily involve a more complex set of verification criteria, which may include information such as the density of image features, the actual number of matched model features, the extent (e.g. lengths) of matched features, etc.

Recognition with "Easy" Images

In this section, 208 images were acquired of one of the models in isolation. These images would be appropriate for the intermediate training stage mentioned in Section 3.6.1, and are used in Section 6.3.2 to demonstrate incremental learning. Examples are shown in Figure 4.18 (top panel). The difficulties introduced by moving from synthetic images to simple real images are due to lighting and feature detection. These factors complicate indexing and recognition, as they can give rise to the following image events:

(a) extra edges — from reflections on some object surfaces

(b) missing edges — due to lack of contrast over the extent of an edge

(c) broken edges — from lack of contrast in a local area, or due to a segmentation error

(d) combined edges — two distinct edges erroneously linked by the segmentation algorithm

Since verification may fail in three ways — validating a false positive, rejecting a true positive, or computing the wrong pose for a true positive — we checked the performance of

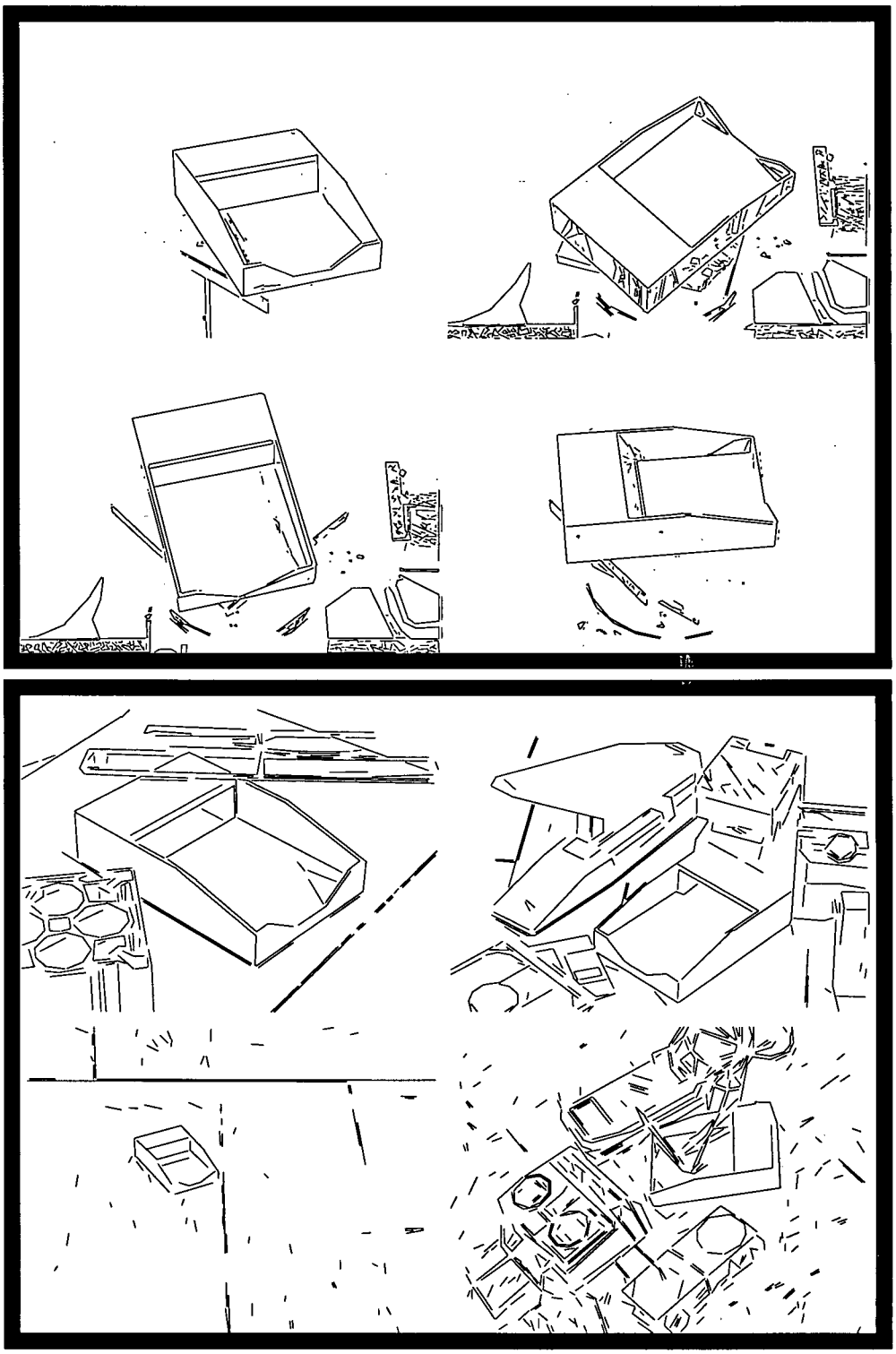


Figure 4.18: Upper panel: Four examples of "easy" images. Lower panel: Four examples of "moderate" images.

the indexing component by hand. Similar to the synthetic image experiments, hypotheses were deemed valid if, in the judgement of the experimenter, the image grouping properly matched the model grouping. Again, this meant that indexing could be deemed successful even if verification failed. It also allowed for a detailed analysis of the failure modes of the system, as the entire recognition process was witnessed for each image.

Table 4.2 gives the results for all three image regimes. "Rank" means the rank of the first valid match hypothesis, for those images where indexing was successful. In the set of 208 "easy" images, 198 were correctly indexed using only 4-segment chain groupings (95%). Of the 10 that were not, all were "degenerate" views, taken directly from the front or the side of the object. By inspection, it was clear that a human would also have had trouble identifying the object in these particular images. Nevertheless, when 3-segment chains and 3-segment parallel groupings were included, the number of successful index attempts climbed to 207 out of 208.

From the over 200,000 training vectors, the algorithm was able to provide a good match after only 2.7 verifications, on average. In fact, the majority of times (126 of 207) a valid hypothesis was ranked first, and only in a few cases (six) was there a need to explore more than 10 hypotheses before the object was found.

Of the 207 successful index attempts, 47 verifications failed. However, of these, 25 failed due to the 2-fold degeneracy problem of planar feature groupings (Section 4.5.2). This is easily correctable by trying both possible orientations (either tilted towards or away from the image plane). 11 failed because of erroneous initial pose estimates, stemming from inaccuracies in image feature positions. However, when the system was allowed to continue searching the hypothesis list after these failures (which would have been the case for an automatic recognition system), indexing and verification eventually succeeded in 8 of the 11 cases, by using a different (but still highly-ranked) image grouping for indexing and pose initiation.

A further 9 of the verification failures actually signalled recognition properly, but the final pose determination was slightly off, due to an incorrect choice of matches between two closely-spaced parallel segments. As stated earlier, a mechanism to detect and avoid match

decisions when some ambiguity exists would relieve this problem. The remaining 2 verifications failed due to a combination of reasons.

Recognition with "Moderate" Images

For these experiments, 25 images were gathered of a single database object in the presence of significant clutter and noise, variable amounts of occlusion, and with the object appearing in various sizes. Performance trends were similar to the "easy" image set, but slightly worse in all categories. The most significant drop was in the percent found using just the 4-segment chain groupings. This was due to several factors — image noise, object size, occlusions — conspiring to make longer chains less likely to be detected. Ranking was again shown to be very important since, in the majority of cases, only a few verifications were required before a valid hypothesis was discovered. The differences between the average and median rankings show that there were a few "outlier" cases, where a large number of verifications were necessary, but for the most part the selectivity was quite good.

All but two of the valid indices led to proper verifications. These results are perhaps surprising: it might be expected that verification would perform poorly with clutter, since a poor initial pose estimate can lead to erroneous matches. A close examination of the verification process revealed that a large part of this success was due to the match extension approach described in Section 4.5.2. The two successful indices that were not properly verified corresponded to the third failure mode mentioned above, correctly signalling object presence, but providing an inaccurate final pose.

Recognition with "Difficult" Images

This image set consists of 16 images: 4 images of each of 4 database objects. The field of view is larger than for the previous sets, so objects and features are generally smaller. Image noise is therefore able to disrupt the indexing mechanism more frequently. Occlusion is also a significant factor in many of these images, which are more complex than those analyzed in

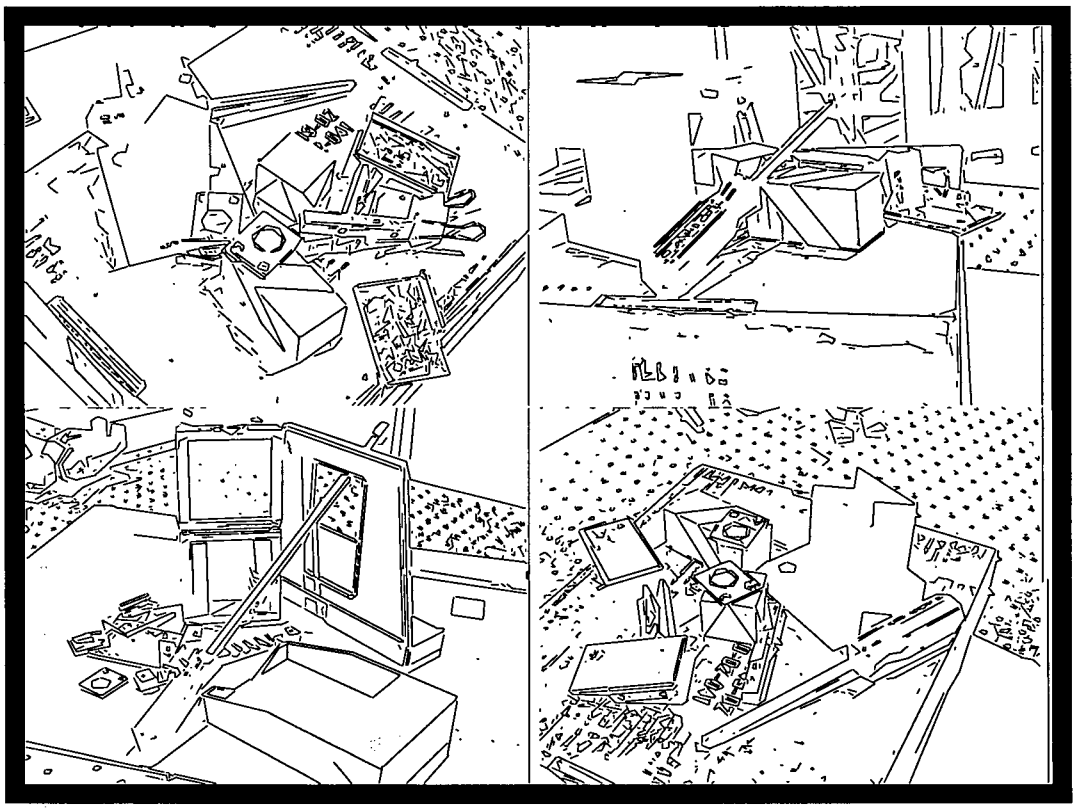


Figure 4.19: Examples of "difficult" images.

most previous work on indexing.

Performance is relatively poor, although adding more grouping types improves things significantly, and adding even more configurations would likely lead to further improvement. Of the 8 successful indexing attempts, 7 were properly verified, and the remaining case failed due to the 2-fold degeneracy problem of planar feature sets. The relatively large value for average rank indicates that the high level of noise and clutter caused problems in these images, but the median rank demonstrates that probability estimates are still worthwhile, as in most cases no more than 50 of the 1000 hypotheses had to be tested before recognition succeeded.

Summary

In this section we demonstrated strong indexing performance on sets of images that we have qualitatively described as "easy" to "moderately difficult" for recognition. Performance degrades for the set of images we have labelled "difficult". Importantly, when recognition fails, it is not the indexing mechanism *per se* that fails, but feature detection and grouping, or verification. This is not to claim that these other problems are trivial, but rather to argue that if feature-based recognition is the way to do object recognition, then indexing functions are the way to do feature-based recognition.

To improve indexing performance, a richer set of feature groupings are required. As well, feature detection will require some engineering. For example, since corners are extremely important for chain groupings, it may be better to use an algorithm designed specifically to detect corner features, rather than grouping straight edge segments together after they have been segmented from the image. As well, the overall recognition procedure can be improved by the two straightforward additions to the verification procedure already mentioned, to deal with 2-fold degeneracy of planar groupings and handling ambiguous matches.

Indexing was shown to work extremely well using only crude choices for the parameters (k, σ) and small, simple grouping types. Perhaps the most important untested attribute of the system is how it will scale with the number of stored models. A larger database will likely require larger groupings for discrimination, although the specificity of 4-segment chains with respect to the current database provides hope that extremely large groupings will not be required. Aspects of scaling are dealt with in the next two chapters and suggest that the approach is quite promising for use with a larger number of models. In Chapter 5, we investigate nearest neighbour lookup in high-dimensional spaces and with large numbers of stored points, and in Chapter 6, we examine the problem of how to add new data into the index to improve probability estimates while at the same time avoiding any increase in the storage requirements.

4.7 Pseudo Real-time System

In this section, we summarize the results of Lloyd et al. (1997), in which a pseudo real-time version of the indexing and object recognition code was implemented and used in a model-based telerobotics experiment to perform robotic assembly-like tasks over the internet. The vision system was used there not only to determine object presence, but also to provide accurate positional information, so that the robotic arm was able to "pick-and-place" the recognized objects.

Telerobotics is robotics in which the physical manipulation of the objects takes place at a site that is remote from the operator of the robot. This may be because the site presents certain hazards to human health, for example. *Model-based telerobotics* is a framework in which the operator interacts with a model of the remote site, rather than with the remote site directly. (Note that this use of the term "model-based" is entirely seperate from its earlier usage in the phrase "model-based object recognition".) There are several advantages to this approach, which include (a) overcoming problems with time delays and/or limited bandwidth, which can cause actions at the remote site to be perceived at a later time than they actually occur, (b) the ability to specify tasks at a higher semantic level, (c) operator control of viewpoint, using a graphical rendering of the model, and (d) the ability to preview actions at the local site before sending the commands on to the remote site.

In this project, the vision system was used to initialize and update a model of the remote site. Figure 4.20 shows the setup of this site. The robotic task which was considered was that of repositioning rectangular blocks, chosen in order to simplify grasping and contact operations rather than as an aid to the vision system. In fact, recognition is somewhat more difficult with such simple objects, as they may not generate as many unique features as more complex objects. The camera observed the work area, recognizing objects (in this case, the two different types of block) and computing pose for each one detected. After each cycle of recognition, which included edge detection, grouping, indexing, and verification, the object labels and pose

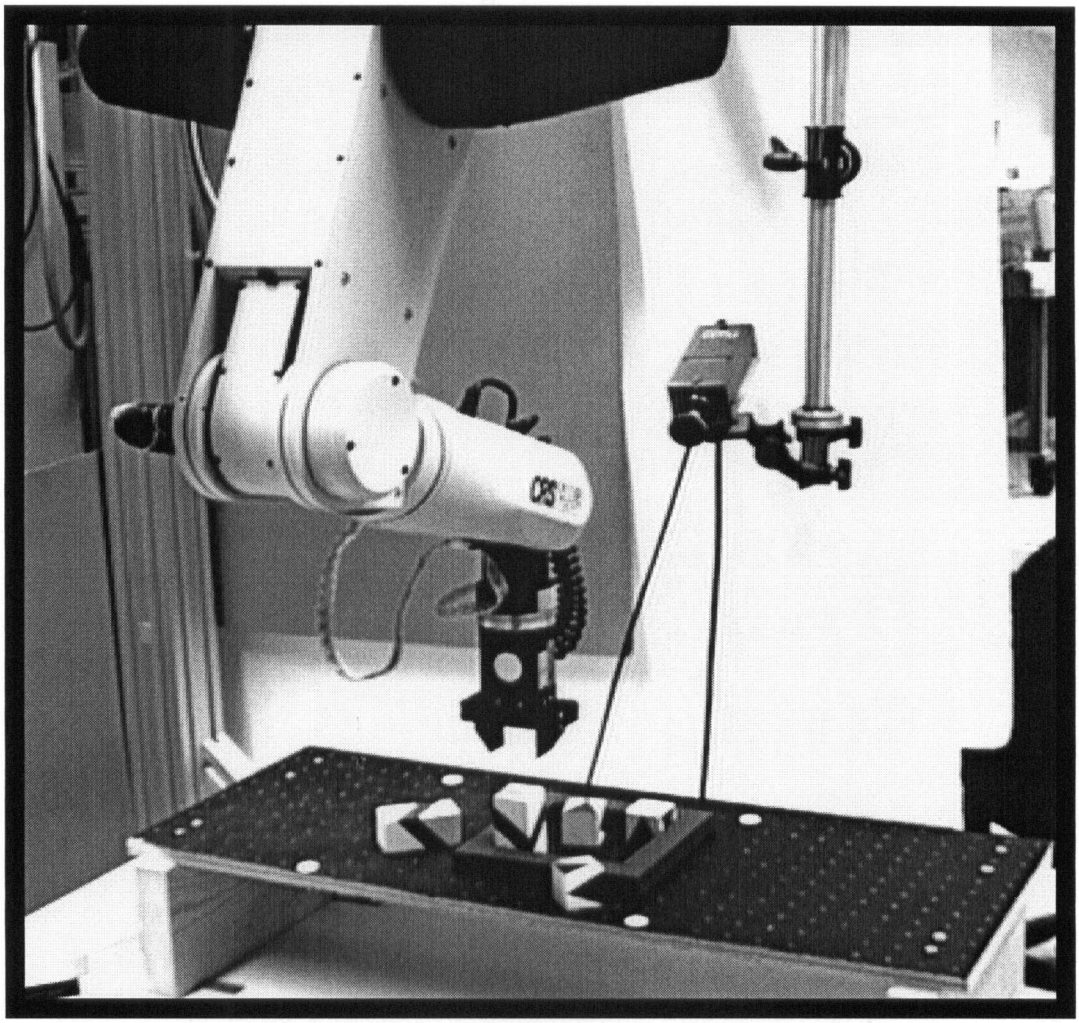


Figure 4.20: Remote site, showing the robot, camera, and work area.

information were sent from the remote to the local site. In cases of low bandwidth, this entire process would ideally take far less time than sending one frame of video for the scene.

At the local site, a GUI (Figure 4.21) allowed the operator to manipulate a model of the scene, which included the table top and the blocks positioned according to the information from the recognition system. Tasks were specified using only a mouse to interact with the model. Once the operator verified that the new block position was correct within the manipulation window, a task command consisting of the block label and new block position was sent to the

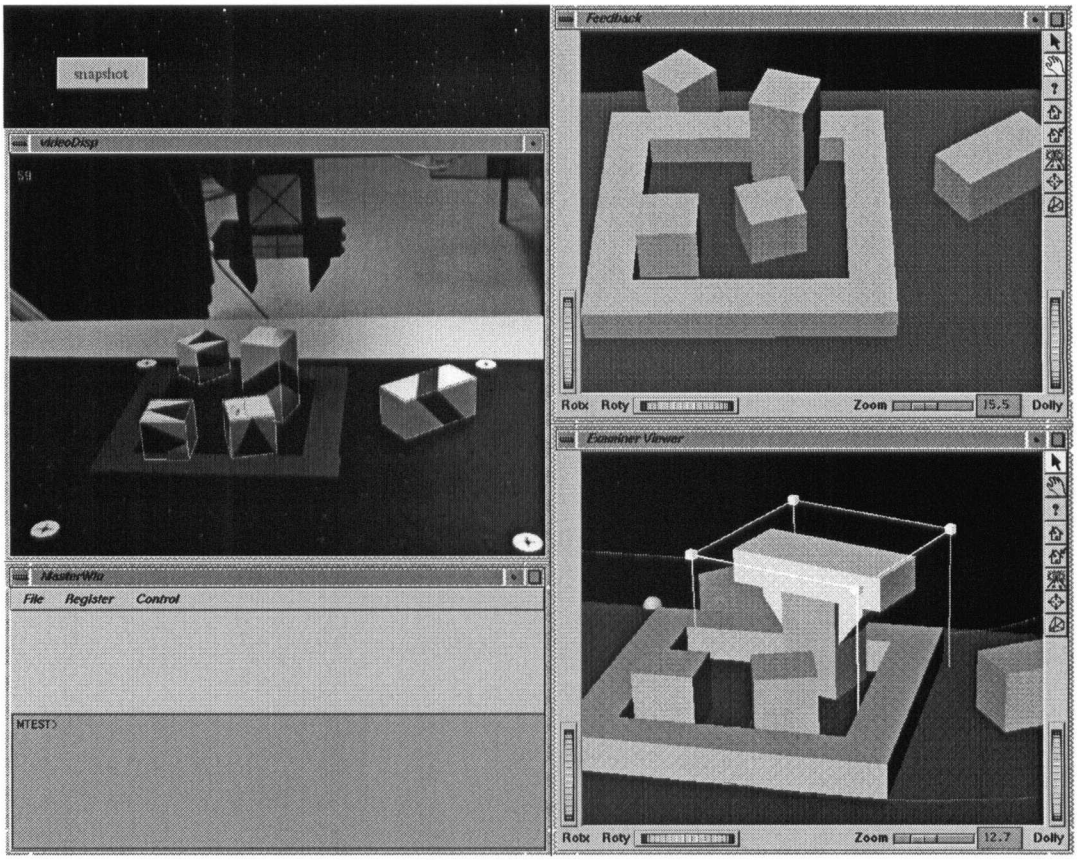


Figure 4.21: Screen image at operator (local) site. Left window: Camera image window. Upper right window: Feedback model window, showing the latest model of the remote site, as given by the vision module. Lower right window: Object manipulation window. This is periodically initialized to the state of the feedback model window, but then is changed according to operator interactions.

remote site. There, all individual robotic motions, inverse kinematics, motion interpolation, etc., were carried out by a *task controller*, implemented using the Robotic Control C Library (RCCL) (Lloyd & Hayward 1992).

The implementation of the vision system required about 5 seconds per frame for recognition. This included a fast software version of the Canny edge detector (Canny 1986), which took 2-3 seconds per frame, and which could easily be accelerated by the use of specialized hardware. The camera position relative to the workspace was calibrated by having the operator manually identify workspace features of known position within the camera image. One early problem that was noticed was that pose was far more precise in the directions parallel to

the image plane than towards and away from the camera. Accuracy was improved using the constraint that objects close to the table top must in fact lie on the workspace surface. To give some idea of the precision required for the pose, the camera was at a distance of roughly 1 m from the work space, and the gripper had a clearance of only about 0.5 cm on each side of a block.

A second problem was that of false positives, in part due to the simplicity of the object models. A more sophisticated verification procedure would be an important addition to the system, but for this demonstration extra constraints were used instead, and verified object instances outside the physical range of the robotic arm were removed from consideration.

The system was successfully demonstrated at the 1996 PRECARN meeting in Montreal. In that experiment, the operator site was situated at the conference location, the remote site was in Vancouver, and communication was via the internet. Meeting attendees were invited to use the system for the repositioning task, in which about four or five blocks of the type shown in Figures 4.20 and 4.21 were typically present at any given session. Between operator sessions, the blocks were rearranged randomly by an attendant at the remote site.

Novice operators found the task interface quite easy and intuitive to use. There were occasional questions concerning the contents of the feedback model window compared with the camera image window, largely because the information in these two windows arrives at different rates. In some cases, operators moved blocks to a part of the workspace outside the field of view of the camera, relying solely on the model to retrieve them, an operation that would have been virtually impossible by direct manipulation.

The experience gained from observing the system in operation suggested that the use of multiple cameras could provide significant improvement in at least two ways. Firstly, the accuracy of pose determination in the direction perpendicular to the image plane of each camera would be greatly enhanced. Secondly, false positives would be reduced, and overall robustness increased, by requiring consistency between the different camera views. The additional effort required for calibration would be well worth the performance gains, and as cameras are

inexpensive, it may pay to use several (rather than limiting the increase to stereo vision).

Chapter 5

Approximate Nearest Neighbour Search in High-Dimensional Spaces

5.1 Introduction

The goal of indexing is to recover from the index the most similar model shapes to a given image shape. In terms of feature vectors, or points in a feature space, this corresponds to finding a set of *nearest neighbours* (NN) to a query point. Almost all previous indexing approaches in model-based vision (Clemens & Jacobs 1991, Forsyth et al. 1990, Lamdan et al. 1990, Lamdan & Wolfson 1988, Rothwell et al. 1992, Stein & Medioni 1992a, Stein & Medioni 1992b) have used hash tables for this task. This is somewhat surprising since it is well-known in other communities (e.g. pattern recognition, algorithms) that tree structures do the job much more efficiently.

In large part this oversight can be explained by the fact that indexing techniques are generally applied in low-dimensional spaces, where hash table search can be quite efficient and easy to implement. Such spaces are adequate when the number of objects is small, but higher-dimensional feature vectors are essential when the model database becomes large, because they provide a much greater degree of discrimination. Unfortunately, nearest neighbour search times

depend exponentially on the dimension of the space.

One data structure that has been used extensively for NN lookup is the kd-tree (Friedman et al. 1977). While the "curse of dimensionality" is also a problem for kd-trees, the effects are not as severe. Hash table inefficiency is mainly due to the fact that bin sizes are fixed, whereas those in a kd-tree are adaptive to the local density of stored points. Thus, in some cases (high-density region), a hashing approach will have to do a long linear search through many points contained in the bin in which the query point lands; in other cases (low-density), an extensive search through adjacent bins may be required before even a single neighbour is encountered. In addition, there is the difficulty of choosing an appropriate bin size.

In this chapter we will demonstrate that kd-tree indexing techniques are feasible in spaces with up to 10 or 20 dimensions. The combination of highly specific feature vectors and probabilistic hypothesis generation provides extremely efficient indexing. Because the ambiguity of hypotheses can be so reduced, the pose clustering stage used by some methods is not required even for large databases, and any false positives are effectively removed by the over-constrained verification procedure.

Our method relies on the rapid recovery of nearest neighbours from the index. In the worst case, in high-dimensional spaces, standard kd-tree search can be very poor, having to examine a large fraction of the points in the space to find the exact nearest neighbour. However, a variant of this search which efficiently finds *approximate* neighbours will be used to limit the search time. The algorithm, which we have called *Best Bin First* (BBF) search (Beis & Lowe 1997), finds the nearest neighbour for a large fraction of queries, and finds a very good neighbour the remaining times. This type of search clearly has wider application than for shape indexing alone, and will be useful for any task where speed is more critical than exactitude when looking for NN in a large set of stored points. Another vision-related usage would be for closest point matching in "appearance-based recognition" (see e.g. (Nene & Nayar 1996)).

5.2 Previous Work

Here we revisit several prominent indexing methods which have used hash tables for indexing, paying particular attention to the index spaces to which they have been applied. Forsyth et al. (1990) outlined several types of projective invariant features for indexing planar objects viewed in arbitrary $3D$ orientations. However, their experiments were carried out using a $2D$ index space generated by pairs of conics. Rothwell et al. (1992) used $4D$ indices defined by area moments of planar curves, that were first transformed into canonical reference frames. Clemens & Jacobs (1991) generated $4D$ and $6D$ index spaces from hand-grouped point sets. For each of the methods above, the dimensionality of the spaces and the number of models were too low for the inefficiency of hashing to be critical.

The *geometric hashing* indexing technique (e.g. (Lamdan & Wolfson 1988)) is also generally applied in low- (2- or 3-) dimensional index spaces. While this technique differs substantially from other indexing methods in that voting overcomes some of the difficulty with bin boundaries, Grimson & Huttenlocher (1990) notes that performance is poor even with small amounts of noise and clutter. This is due to the hash table becoming overloaded with entries, and indicates that the use of higher-dimensional spaces is important. In (Lamdan et al. 1990), geometric hashing is applied with larger ($8D$) indices generated from planar curves ("footprints"). However, the experiments there did not truly test indexing performance, because only a few models were used, with 3 or 4 features each. Based on experiments presented below for hashing, we surmise that this method would have problems locating neighbours without extensive search.

Stein and Medioni presented a method for $2D$-from-$2D$ (1992a) and $3D$-from-$3D$ (1992b) indexing that also used hash table lookup. While the former method used index spaces of between 3 and 6 dimensions, the latter work considered higher- (up to about 14-) dimensional spaces. They avoided the exponential (with dimension) search for NN by using extremely coarse quantization of the bins (60° for angle features in some cases), and looking only in the single

bin containing the query point. This leads to the recovery of a large number of hypotheses, with a low signal-to-noise ratio, since highly dissimilar shapes are allowed to match. Nor does it preclude missing a significant percentage of good hypotheses which lie just across one of the many bin boundaries.

Califano & Mohan (1994) argue for the use of higher-dimensional spaces. Their analysis indicates a dramatic speedup from increasing the size of the feature vectors used for indexing. However, they again use hash tables for the lookup, and only access a single bin per query. Thus, in their method a pose clustering stage is required to accumulate results, presumably because of the high likelihood that a query will land in a different bin than that of the best neighbour.

In the pattern classification literature, various types of tree have been used to find nearest neighbours. Just as in hashing, these methods divide the space into bins; unlike hashing, the methods are *adaptive* in that the partition of the data space depends on the data points to be stored.

The best-known and most widely used of these is the *kd-tree* (Friedman et al. 1977), a complete binary tree having smaller bins in the higher-density regions of the space. The analysis in the original paper shows expected logarithmic time lookup, but experiments in higher dimensions (Sproull 1991) have shown that the method often suffers from inefficient search, in many cases having to examine a large fraction of the stored points to determine the exact nearest neighbour. Other trees specialized to fast NN lookup exist (e.g. (Kim & Park 1986, Neimann & Goppert 1988)), but are sub-optimal because more constraints are set on where bin boundaries can be placed.

Instead of finding the exact nearest neighbour, if we are willing to accept an *approximate* neighbour in some fraction of the cases, then processing time can be reduced. One early method (Miclet & Dabouz 1983) uses a hierarchical tree data structure in which internal nodes are the centers of mass of the nodes at the next lower level. The point recovered from the first leaf node encountered provides an approximate NN in a very short search time, but the quality of this

neighbour is relatively low. Taking the best neighbour from consecutive searches of three trees containing the same data (but with different hierarchical structures) improves results somewhat.

In (Arya 1995), Arya presents an approach based on neighbourhood graphs. While his main results concern the asymptotic behavior of one particular algorithm, which contains impractically large constant factors, he does give a practical variant called RNG* that demonstrates good results for moderate dimensionalities (8 - 16). Independent of our own work, he also develops the equivalent of BBF search which he terms "priority kd-tree search". Comparisons between RNG* and the priority algorithm show comparable performance for moderate dimensionalities.

Most recently, Nene & Nayar (1996) have used a different type of approximation to NN lookup. They guarantee recovery of the best neighbour only if it is within ϵ of the query point, by using a set of sorted lists of the coordinates of the stored points, and successively eliminating points further away than ϵ, one dimension at a time. This method is effective if ϵ can be kept small (≤ 0.25 where each dimension has unit extent). In higher-dimensional spaces that is only possible when the number of points is extremely large, on the order of $(\frac{1}{\epsilon})^k$. Indeed, their experiments with a uniform distribution in high-dimensional spaces indicate a speedup of only a factor of two over linear-time search through all data points.

5.3 kd-tree Lookup of Nearest Neighbours

Before introducing the BBF search algorithm, we first review the standard version of the kd-tree, with emphasis on why the method may be inefficient for NN lookup. The tree is built as follows (refer to Figure 5.1. Beginning with a complete set of N points in $\Re^k$, the data space is split on the dimension i in which the data exhibits the greatest variance. A cut is made at the median value m of the data in that dimension, so that an equal number of points fall to one side or the other. An internal node is created to store i and m, and the process iterates with both halves of the data. This creates a balanced binary tree with depth $d = \lceil \log_2 N \rceil$.

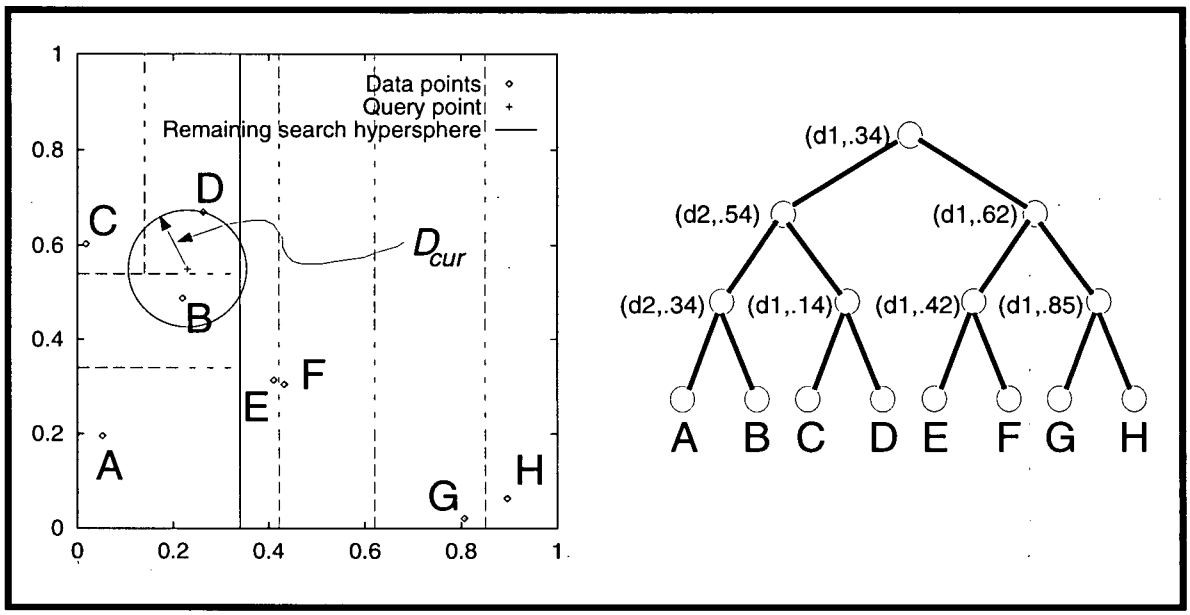


Figure 5.1: kd-tree with 8 data points labelled A-H, dimension of space k=2. On the right is the full tree, the leaf nodes containing the data points. Internal node information consists of the dimension of the cut plane and the value of the cut in that dimension. On the left is the 2D feature space carved into various sizes and shapes of bin, according to the distribution of the data points. The two representations are isomorphic. The situation shown is after initial tree traversal to locate the bin for query point "+" (contains point D). In standard search, the closest nodes in the tree are examined first (starting at C). In BBF search, the closest bins to query point q are examined first (starting at B). The latter is more likely to maximize the overlap of (i) the hypersphere centered on q with radius D_{cur}, and (ii) the hyperrectangle of the bin to be searched. In this case, BBF search reduces the number of leaves to examine, since once point B is discovered, all other branches can be pruned. The standard search would first look at C, and only arrive at B after a second backtracking step.

The leaves of a kd-tree form a complete partition of the data space, with the interesting property that bins are smaller in higher-density regions and larger in lower density areas. This means that there is never an undue accumulation of points in any single bin, and that the NN to any query should lie, with high probability, in the bin where the query falls, or in an adjacent bin.

To look up the NN to a query point q, a backtracking, branch-and-bound search is used. First the tree is traversed to find the bin containing the query point. This requires only d scalar comparisons, and in general the point recovered from the bin is a good approximation to the nearest neighbour. In the backtracking stage, whole branches of the tree can be pruned if the region of space they represent is further from the query point than D_{cur} (the distance from q

BBF search algorithm:

STEP 1: Start at root node. Initialize D_{cur}, the distance from the query point to the current closest neighbour visited, to infinity. Set E_{max}, the maximum number of leaf nodes to visit.

STEP 2: Traverse the tree until reaching a leaf node. At each internal node, follow the branch closest to the query point q. If the other branch is not further from q than D_{cur}, *insert* the branch into a priority queue according to its distance from q.

STEP 3: Check the distance D between the point stored in the leaf node and q. If $D < D_{cur}$, update D_{cur}.

STEP 4: If the number of leaf nodes visited is equal to E_{max}, then the search is terminated.

STEP 5: Otherwise, consult the priority queue. If the queue is empty, or the top entry is further from q than D_{cur}, the search is complete (and in fact the exact nearest neighbour has been found). Otherwise, *pop* the queue, restart the search at the branch that was stored in the top queue entry, and goto STEP 2.

Figure 5.2: BBF search algorithm.

to the closest neighbour yet seen). Search terminates when all unexplored branches have been pruned.

This process can be very effective in low-dimensional spaces, but in higher dimensions there are many more bins adjacent to the central one that must be examined, and performance degrades rapidly. Interestingly, a great deal of this search is spent examining bins in which only a small fraction of their volume could possibly supply the nearest neighbour. If we are willing to settle for an *approximate* NN, then we can avoid prolonged search by simply limiting the number of leaf nodes we are willing to examine (to E_{max}), and settling for the best neighbour

found up to that point.

This modification extends the domain of kd-trees for fast NN recovery a small amount. However, the backtracking search described above is still inefficient because the order of examining leaf nodes is according to the tree structure, which depends only on the stored points, and does not take into account the position of the query point (see Figure 5.1). A simple idea for a more optimal search is to look in bins in order of increasing distance from the query point. The distance to a bin is defined to be the minimum distance between q and any point on the bin boundary.

Such a search can be easily implemented with a small amount of overhead using a priority queue (Figure 5.2). This is a data structure that maintains a sorted list, where insertions and extractions can be accomplished in logarithmic time (by storing it as a heap). During NN lookup, when a decision is made at an internal node to branch in one direction, an entry is added to the priority queue holding information about the option not taken. This includes the current tree position and the distance of the query point from the node. After a leaf node has been examined, the top entry in the priority queue is removed and used to continue the search at the branch containing the next closest bin.

This *Best Bin First* (BBF) search strategy provides a dramatic improvement in the approximate NN search for moderate dimensionality, making indexing in these regimes practical. For fairly small values of E_{max}, the method discovers the exact NN a large percentage of the time, and a very close neighbour in the remaining cases. The next section provides experimental evidence to support this claim. Note that, with BBF search we can guarantee an exact query just as with standard kd-tree search, by allowing an unlimited number of leaf nodes to be examined (i.e. setting E_{max} greater than the number of stored points). The exact neighbour will be discovered because, as in the standard search, all branches are visited unless they have been pruned, and a branch is only pruned if all points represented by the branch are *further away than the current closest neighbour seen.* This property may be important in other uses of the BBF algorithm, for example in database queries, where in some cases an approximate

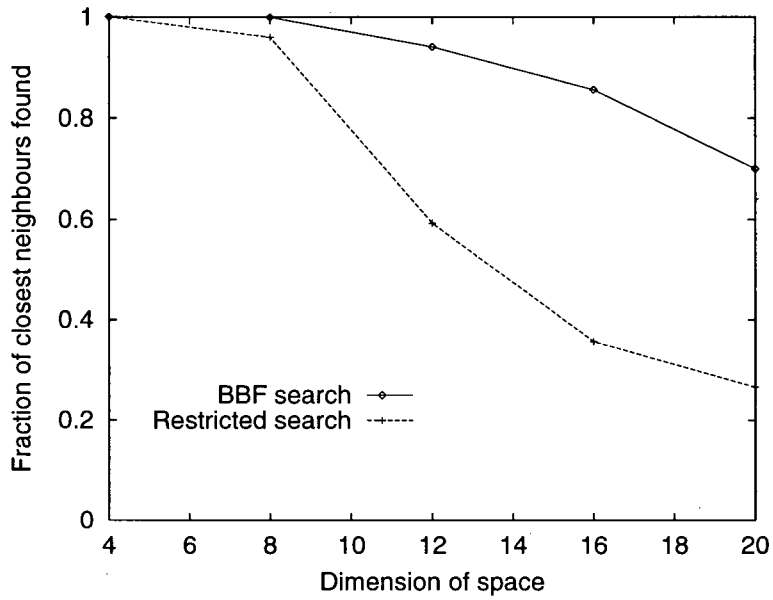


Figure 5.3: Approximate NN lookup vs dimension of space: Fraction of neighbours found. (Uniform distribution; 100,000 stored points; E_{max}(BBF) = 200; E_{max}(restricted) = 480; averaged over 1000 queries.)

solution is appropriate but in other, higher priority queries, an exact solution is critical.

5.4 Experimental Results

5.4.1 Uniform Distribution

We perform experiments with points randomly drawn from a uniform distributed in the unit hypercube, and present results for the 1-NN problem, which are also indicative of the performance of the methods for k-NN lookup. Rather than providing complete coverage of the three variables of interest (dimension of index space D, the number of stored points N, and E_{max}), in each experiment two of the three are fixed and the other allowed to vary. In each case, changing the fixed values only changes the absolute numbers on the curves, while the trends remain the same as those shown. The values of the parameters when fixed are $D = 12$, $N = 10^5$, and $E_{max} = 200$. This point is on all curves, and we believe it provides a challenging scenario for current indexing systems.

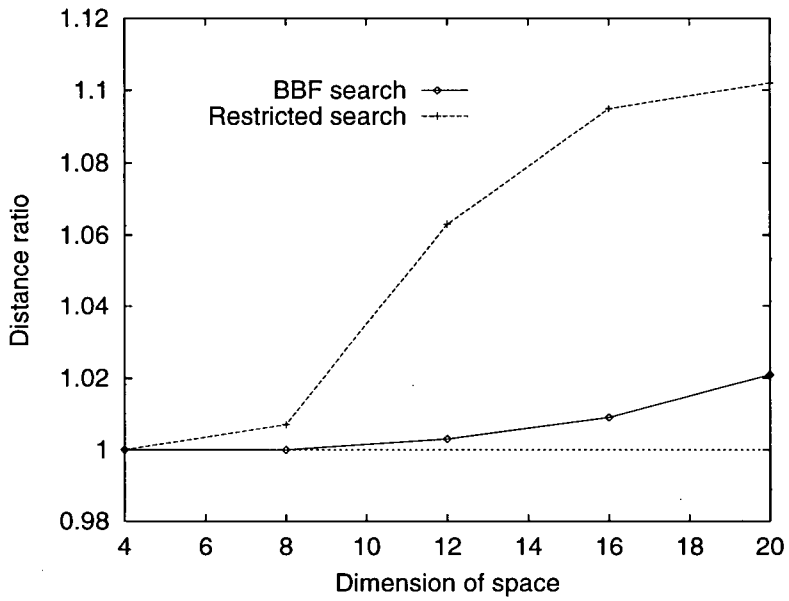


Figure 5.4: Approximate NN lookup vs dimension of space: Average of ratios of approximate to actual NN distance. (Uniform distribution; 100,000 stored points; E_{max}(BBF) = 200; E_{max}(restricted) = 480; averaged over 1000 queries.)

We define "restricted" search to be the standard kd-tree search that is terminated after examining at most E_{max} leaves, and "exact" search to be standard kd-tree search with no restrictions. Because of the overhead involved in keeping the priority queue for BBF search, it is unfair to make a straight comparison between BBF and restricted search. Based on timing experiments which covered the entire range of parameters that were used, the number of leaves that standard kd-tree search is able to visit in the same time that BBF completes its search is a factor of between 1.8 and 2.4 times the number for BBF. Therefore, we allow restricted search to visit 2.4 times the number of leaves that BBF does in the experiments that follow.

Figure 5.3 sketches the performance of approximate search with respect to the dimension of the space. Restricted search rapidly deteriorates above D=8 whereas BBF degrades smoothly. In some sense this curve is the most important to indicate that indexing in moderate dimensionalities is practical. In 12 dimensions, for example, BBF recovers the closest neighbour 94% of the time (versus 59% for restricted search) while on average examining only 200 of the

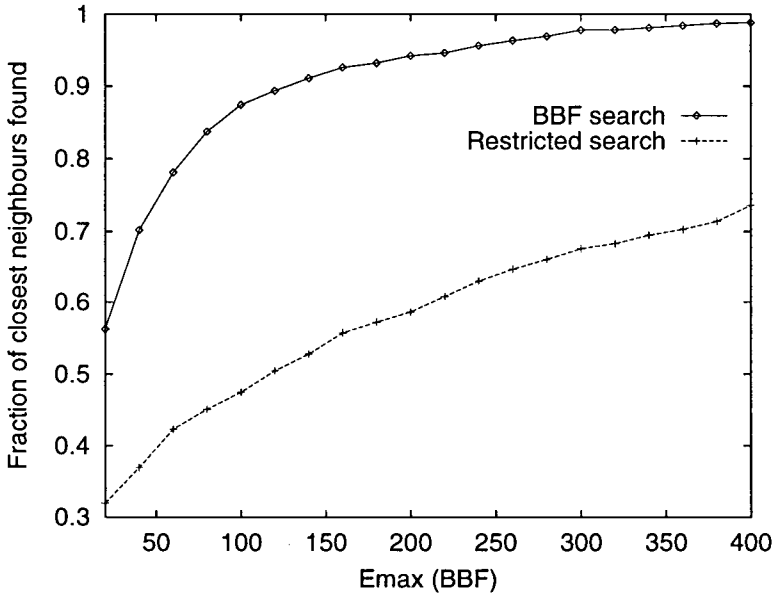


Figure 5.5: Approximate NN lookup vs E_{max}. (Uniform distribution; 100,000 stored points; dimension of space = 12; averaged over 1000 queries. E_{max}(restricted) was 2.4 times E_{max}(BBF).)

100,000 leaf nodes. For this same case, the "exact" search has to examine over 2400 leaves.

Figure 5.4, obtained using the same parameters as Figure 5.3, shows that even when the method does not discover the exact nearest neighbour, it does not fail completely. To the contrary, even up to dimension 20, the average distance of recovered neighbours is *only* 2% *greater* than the true NN distances. This is important when considering the method of indexing proposed in this thesis, which uses the distance-weighted support of several neighbours to compute probability estimates. Even if the exact NN is not among this set, it is likely that some other close neighbours, corresponding to other nearby training views of the same object, will contribute to the correct classification (i.e., match hypothesis).

Figure 5.5 gives performance with respect to E_{max}. For this graph, the x-axis values are for BBF only, and the values for restricted search are equal to the BBF values multiplied by 2.4. For reasonable values of E_{max} (150-400) the closest neighbour is found by BBF more than 90% of the time.

Figure 5.6 shows that performance of BBF varies remarkably slowly with the number

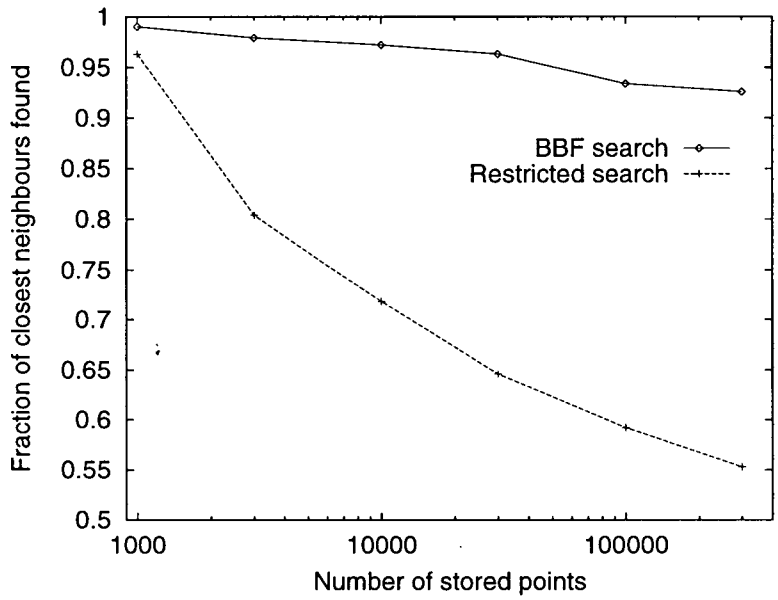


Figure 5.6: Approximate NN lookup vs Number of Stored Points. (Uniform distribution; E_{max}(BBF) = 200; E_{max}(restricted) = 480; dimension of space = 12; averaged over 1000 queries.)

of stored points. In this $12D$ space, the algorithm uncovers better than 92% of the exact neighbours for up to 300,000 points. In the latter case, BBF examines only one out of every 1500 points, whereas the exact kd-tree search must examine one out of every 100 points to ensure that the closest neighbour is recovered.

In order to show that the overhead involved in BBF search is not large, we performed some timing experiments in which we compared the algorithm both to exhaustive search and to the exact kd-tree search algorithm (Figure 5.7). In dimension 10, for example, BBF provides speedup by a factor of 60 over exhaustive search, and in dimension 20 it is still an order of magnitude faster. This graph compares favorably with a similar one given in (Nene & Nayar 1996) for their approximate NN algorithm. They show a speedup by a factor of about two for dimensions 15 and above.

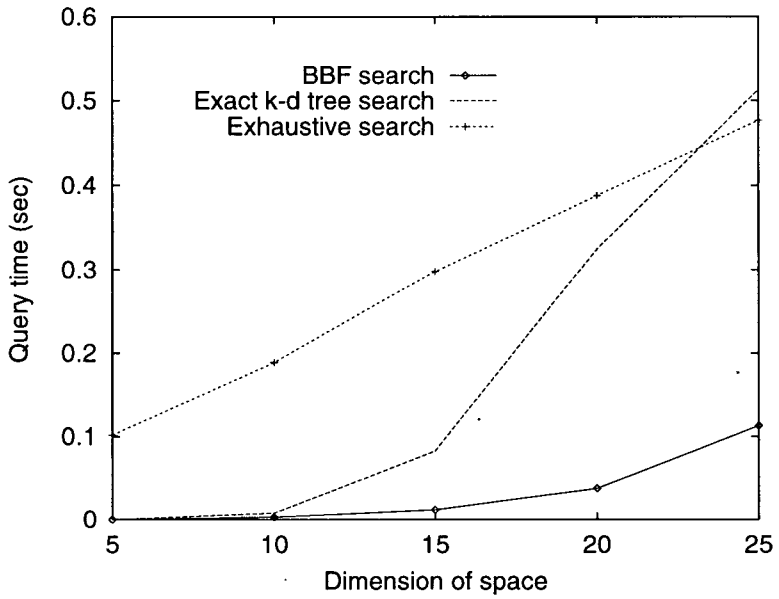


Figure 5.7: Timing comparison of NN lookup. (Uniform distribution; 30,000 stored points; E_{max}(BBF) set to recover 95% of exact neighbours; Averaged over 1000 queries.)

5.4.2 Hash Table Lookup

While the above experiments demonstrate the advantages of BBF search over restricted kd-tree search, it is important to note that the standard kd-tree search algorithm is already a significant improvement over hash table lookup. In this section we compare with the hashing method used in (Califano & Mohan 1994, Stein & Medioni 1992a, Stein & Medioni 1992b), in which only a single bin is accessed for each query. This is clearly another form of approximate NN search, but one which performs very poorly in high-dimensional spaces. More extensive hash searches that would perform better are infeasible: searching to either side of the query bin in each dimension would mean accessing 3^k bins; and the computations required to do a more intelligent search, similar to BBF (i.e., to determine the next closest hash bin to check during a search of adjacent bins) are too expensive.

For this "single bin" hash lookup we have chosen a situation quite favorable to the method: the dimension of the space is not extreme ($8D$); the distribution of points is uniform

$\frac{\#bins}{dim}$	3	4	5
Fraction NN found	0.318	0.177	0.072

Table 5.1: Hash table lookup: fraction of nearest neighbours found in "single bin" access. (Uniform distribution; 62,536 stored points; dimension of space = 8; averaged over 1000 queries.)

(best case for hashing), in the unit hypercube; and the bin sizes are relatively large (as suggested by Califano & Mohan (1994) and Stein & Medioni (1992b)). The number of points ($4^8 = 62,536$) is chosen to give $1\frac{point}{bin}$ in the moderate-density regime, when $\frac{\#bins}{dim} = 4$. This leads to $10\frac{points}{bin}$ when $\frac{\#bins}{dim} = 3$ ("high-density"), and to $0.17\frac{points}{bin}$ when $\frac{\#bins}{dim} = 5$ ("low-density"). Table 5.1 shows that, no matter how coarsely the bins are quantized, the method performs poorly. As a comparison, for the same problem BBF requires $E_{max} \approx 60$ to recover an average of 95% of the exact neighbours.

As the dimensionality increases, the numbers for hashing become even worse very quickly, for two reasons. First, there is the inevitable decline due to the fact that a larger fraction of the points become potential nearest neighbours. (This is the same problem that causes the standard kd-tree to become impractical in higher dimensions.) Secondly, unless the number of points is increased exponentially with dimension, even with the coarsest possible quantization ($2\frac{bins}{dim}$) the average bin will be very unlikely to contain any points. Then the performance will look more like the low-density entry of Table 5.1 than either of the others.

5.4.3 Synthetic Model Database

We also performed an experiment using non-uniform data, provided by our database of object models. From each of the 10 models we extracted 10,000 feature vectors by taking synthetic images from random viewpoints, which required an average of 103 images per object. The feature groupings we used were 6-segment chains, which provided a $10D$ feature space consisting of 5 angles and 5 edge-length ratios.

Recall from previous chapters that in our indexing method, rather than using a sin-

gle NN, hypotheses are formed using a distance-weighted combination of a small number of recovered neighbours, in this case 10. Hypotheses are then sorted into a list, based on the weightings. In this experiment we compare the "approximate" lists generated by BBF, with a very small value for E_{max} (50), to the "true" lists, as determined by the exact algorithm. Results were averaged over 1000 test images, each one comprised of a single database model chosen at random, and viewed from a random viewpoint. The average length of a "true list" generated from an image was 93 hypotheses.

The speedup from using BBF over exact kd-tree search was a factor of 14. Even though only 50 of the 100,000 stored points were being examined, BBF missed less than 4% of the hypotheses, and even more significantly, the average ranking of the first missing hypothesis was 80^{th} out of 93. As well, the average difference in the rank of a hypothesis between the lists was less than 2 positions. As mentioned earlier, these results are partially due to the fact that most of the good hypotheses have the support of a few neighbours (which correspond to the same grouping as seen from nearby viewpoints in the training set), so if only one of these is missing, indexing is still able to succeed. BBF search discovered 96.2% of the objects, and the average rank of the successful index was 2.7.

5.5 Summary

High-dimensional feature spaces provide a degree of discrimination that is essential for large model databases, which is the natural domain of application for indexing techniques. Our experiments demonstrated that BBF search is capable of finding close neighbours over a wide range of dimensions and for very large numbers of stored points, by examining only a small fraction of the points. We also presented some results indicating that the hash table method most commonly used for indexing breaks down in higher dimensions.

The BBF method would also be applicable to other areas such as appearance-based recognition. There, objects are represented as low-dimensional manifolds embedded in high-

dimensional spaces, and stored as point sets sampled from the manifolds. Object classification is achieved using NN lookup in the large-dimensional spaces, and BBF appears to be more efficient than the methods currently used for this task (e.g. (Nene & Nayar 1996)).

Chapter 6

Incremental Learning and Clustering

6.1 Introduction

The current indexing system is able to achieve a certain level of performance by training strictly with the synthetic models, according to the assumption of equal *a priori* probabilities for any viewpoint. Earlier this was referred to as "bootstrap" learning. As described in Section 3.6.1, this can be enhanced by adding an *incremental learning* stage to the system. There, data from images where recognition has occurred can be added to the index, to update and refine the probabilities that are in effect stored there. Improvements to the probability estimates will lead to faster recognition times, since the hypothesis rankings which are based on the estimates will in turn become more accurate.

Probabilities improve in two senses. Firstly, *a priori* probabilities will be changed according to which objects and which views of objects appear more than others. Secondly, the distribution of samples from each individual 3D feature grouping will be modified, according to how that grouping appears in real images given the effects of lighting, feature detection, etc. In this chapter, experiments will be presented which demonstrate improvement in indexing performance from incremental learning, from both kinds of probability update. The first type is much easier to investigate, since synthetic data can be used to vary *a priori* probabilities in

several ways. The second type unfortunately requires a large amount of real data: while the random shifting of feature values might be approximated by some noise model, predicting the systematic variation in detectability of features, for example due to interaction of light with the object surface, or perhaps due to an image processing algorithm, is not really feasible.

So that the system may eventually reach optimum performance, we imagine that it is constantly engaged in incremental learning. This scenario raises the issue of finite resources, however, as the system is continually incorporating into the index information gained while the system is in operation. There are potential problems both with the physical size of the memory, and with the increase in index access time with index size (although as we have seen in the previous chapter, the latter is relatively slow). One way to deal with the problems of increased memory is to use a *clustering* method to reduce the number of stored points. At the same time, the distortion to the probability distributions implied by the pre- and post-clustered point sets must be minimized, as these are the same distributions required to provide accurate rankings. In this chapter, we look at various approaches to clustering, and present a new algorithm that is simple and effective. Experiments show that performance using the combination of incremental learning and clustering may not be much worse than applying incremental learning with "infinite memory". This suggests that it may be possible to improve performance by learning how objects appear in real images, without increasing the amount of memory required to represent the object.

In what follows, we first present relevant previous work in clustering. This is followed by our own clustering algorithm, which is a hybrid of two recent simple and efficient methods, and provides a slight improvement over both. The algorithm, called "Weighted Vector Quantization" or WVQ, is then applied to some standard test data and evaluated according to its classification performance. Finally, experiments in incremental learning with the model database from Chapter 4 are presented, both with and without the use of clustering, and the indexing performance of both is compared.

6.2 Previous Work

6.2.1 Standard Clustering Methods

Standard clustering methods typically start by fixing the number of prototypes, subsequently applying a two-stage iterative procedure characterized as (Shimoji & Lee 1994):

(a) **Nearest neighbour assignment**, in which each of the original data points is assigned to one (or more) of the prototypes; and

(b) **Centering**, in which a new position is computed for each prototype, according to the latest set of nearest neighbour assignments.

There are two classic techniques for clustering, namely *k-means* clustering and *LBG vector quantization*, which are very similar. Suppose a set of prototypes has somehow been initialized at various locations within the data space. In the NN-assignment step, each data point is simply associated with the closest current prototype. In the centering step, the position of a prototype is taken to be the mean of all data points that are currently associated with it. The only difference between the two methods is the time at which the centering stage is applied. In LBG, the full set of NN assignments are determined, and then the full set of prototype positions are re-computed. In k-means, prototypes are updated each time a single data point switches allegiance from one prototype to another. In both methods, the quality of the final solution is taken to be the sum of the squared distances between the data points and their associated prototypes.

A similar approach known as "fuzzy" clustering (Bezdek 1980) consists of the same two-stage process, but allows for partial NN-assignments of data points to prototypes (with the total weight of assignments for each original point summing to 1). In the centering stage, new prototype positions are computed as a weighted sum of the associated data points.

The above methods have two major problems: (i) How many prototypes should be used? and (ii) How are the prototype positions initialized? For the standard clustering problem, there

appears to be no ideal solution to the first issue, since the whole idea is to pick out from the original data the number of natural clusters which ostensibly exist. In the DRDE problem, the object is to provide minimal distortion to the probability estimates implied by the data, so in general it is better to use as many prototypes as possible (up to the number of original data points). Of course, there is a hard upper limit that might eventually be tested by incremental learning, which is the number of prototypes that fit in the machine's memory.

The second problem is typically solved by setting initial prototype positions to coincide with a random subset of the original data points. There is some probability that this will approximate the true, underlying distribution, so the prototypes will not have to shift very much on average during the clustering iterations. It is fairly important that the starting point of the system is close to a good solution, since all three methods correspond to slightly different versions of a straightforward descent algorithm in the space of possible solutions (with sum-of-squared-differences as the measure of quality). Less satisfying solutions for the initial positions, which will nevertheless work in many situations, are to place the prototypes randomly within the data space, or at equally spaced intervals.

6.2.2 Physics-Like Approaches

Deterministic Annealing

The idea behind this method is to cast clustering as a physical problem and to use a statistical mechanical analysis to determine "equilibrium states" of the system. In the end, Deterministic Annealing (Rose, Gurewitz & Fox 1990, Rose, Gurewitz & Fox 1993) turns out to be essentially a fuzzy clustering technique, with a parameter β ("inverse temperature") that controls the level of fuzziness. Hard clustering solutions are recovered as $\beta \to \infty$.

The analysis begins by defining an energy function which depends only on the system configuration, just as in a real physical system. Supposing a cost function $E_j(\vec{\mathbf{x}})$ exists for

assigning data point $\vec{\mathbf{x}}$ to a cluster C_j, the total expected cost will be given by:

$$\langle E \rangle = \sum_i \sum_j P(\vec{\mathbf{x}}_i \in C_j) E_j(\vec{\mathbf{x}}_i) \tag{6.1}$$

The most non-committal we can be regarding the true data distribution, about which we want to make no assumptions (to form the least-biased estimate), is by applying the principle of maximum entropy (see interesting discussion in (Jaynes 1957)). Maximizing entropy under the energy constraint above leads to

$$P(\vec{\mathbf{x}}_i \in C_j) = e^{-\beta E_j(\vec{\mathbf{x}}_i)} / Z_i \tag{6.2}$$

$$Z_i = \sum_k e^{-\beta E_k(\vec{\mathbf{x}}_i)}. \tag{6.3}$$

After some calculation, and using the squared-distance $E_j(\vec{\mathbf{x}}) \equiv \|\vec{\mathbf{x}} - \vec{\mathbf{y}}_j\|$ as the cost function, where $\vec{\mathbf{y}}_j$ are the prototype positions, it is found that at a minimum of the total expected cost, the following conditions hold:

$$P(\vec{\mathbf{x}}_i \in C_j) = \frac{e^{-\beta \|\vec{\mathbf{x}}_i - \vec{\mathbf{y}}_j\|^2}}{\sum_k e^{-\beta \|\vec{\mathbf{x}}_i - \vec{\mathbf{y}}_k\|^2}} \tag{6.4}$$

$$\vec{\mathbf{y}}_j = \frac{\sum_i \vec{\mathbf{x}}_i P(\vec{\mathbf{x}}_i \in C_j)}{\sum_i P(\vec{\mathbf{x}}_i \in C_j)} \tag{6.5}$$

By observation, these "equilibrium states" correspond to the nearest neighbour and centering stages of the earlier clustering algorithms, and suggest the same iterative optimization procedure. Assignments of data points to prototypes are partial, as in fuzzy clustering, but with a different weighting function. Deterministic annealing reduces to LBG when $\beta \to \infty$ and all probabilities become either zero or one. When $\beta = 1$, DA is equivalent to a simple version of another prominent algorithm known as "Expectation-Maximization", but in that method the *shapes* of the Gaussians are optimized in addition to their positions.

DA requires the number of prototypes to be set just as in the previous iterative approaches. However, at a given temperature it may turn out that some of these are identical, or coalesced, so that a "natural" number of prototypes appears. For example, at $\beta = 0$, all

prototypes are collapsed at the centroid of the sample data. As β is raised, phase transitions are encountered where prototypes split off to form new, smaller clusters, hopefully "tracking" the optimal solution and avoiding local minima. In practice, the number of prototypes is simply initialized to the number of data points, and the problem becomes finding an appropriate annealing schedule. The method is more computationally intensive than others because, for each value of β, a complete set of NN/centering iterations is performed.

Entropic Scheduling

Another interesting method is that of Shimoji & Lee (1994), in which β is again used as a control parameter. The standard 2-stage iterative algorithm is applied, but instead of the deterministic nearest neighbour assignment step of k-means or LBG, a *probabilistic* assignment rule is used. The probability that a data point is associated with a given prototype is inversely related to its distance from that prototype, and depends on β (see Equation 6.4).

Again, as β gets very large, the hard clustering solutions are recovered. For smaller values, the method can be seen as a cross between k-means and simulated annealing, with the idea that it provides better solutions than k-means (using probabilistic assignment to escape from local minima) but at a smaller cost than simulated annealing. Due to the extra complication (and compute time) involved in the annealing process, and the likelihood that a simpler approach will work almost as well, these methods are not considered important for solving the indexing DRDE problem.

6.2.3 Data Reduction for Classification

These methods take a different approach to the data reduction problem, in which the ultimate goal of classification leads to completely different distortion measures. In these algorithms, a reduced set of prototypes is formed by pruning the original data set. The effective probability distributions are explicitly allowed to change, and the distortion measure is simply the classification error, typically using a NN classification rule.

Condensed nearest neighbours (CNN) (Hart 1967) and **Reduced nearest neighbours** (RNN) (Gates 1972) both remove prototypes in regions where there is no ambiguity of classification. Both methods provide a *consistent subset* of the data, which means that all of the original examples are correctly classified (by the NN rule) using the reduced point set. **Edited nearest neighbours** (ENN) (Devijver & Kittler 1980) takes the opposite approach, removing points that are mis-classified by the kNN rule. This process leaves well-separated clusters of points all having the same label. CNN can be applied to the resulting point set for further reduction. This method tacitly assumes that the overlap between classes is relatively small.

These techniques typically provide a small reduction in the number of prototypes, perhaps a factor of 2 or 3. While this type of clustering is reasonable when a yes/no classification decision is to be made, it is inappropriate for our problem, in which more accurate density estimates are required for ranking hypotheses.

6.2.4 Data Reduction for Density Estimation

In these methods, the idea is to reduce the number of prototypes while at the same time maintaining an accurate representation of the underlying probability distribution. Because the true densities will usually not be known, the trick is to first choose an estimator like kNN or Parzen Windows, and then to try to minimize the difference between the pre- and post-clustering estimates. Of course, the estimated densities can be used to perform classification, and this is typically how these techniques are evaluated.

Fukunaga's "reduced Parzen classifier" (Fukunaga & Hayes 1989) uses the Parzen Windows estimate

$$\hat{p}_N(X) = \frac{1}{N}\sum_{i=1}^{N} k(X - X_i) \tag{6.6}$$

where X_i are the initial data points and $k(\cdot)$ is the Parzen kernel, typically a Gaussian. The reduced estimate is given by

$$\hat{p}_r(X) = \frac{1}{r}\sum_{i=1}^{r} k(X - Y_i) \tag{6.7}$$

where Y_i are the prototypes from the reduced set.

The measure of similarity between distributions is taken to be the entropy

$$\int \ln(\frac{\hat{p}_r(X)}{\hat{p}_N(X)})\hat{p}_N(X)dX. \tag{6.8}$$

This is equivalent to the expectation $E[\ln(\frac{\hat{p}_r(X)}{\hat{p}_N(X)})]$ over $\hat{p}_N(X)$. Actually performing the integration is infeasible, but replacing the integration by the sum over the sample set gives

$$J = \frac{1}{N}\sum_{i=1}^{N}\Big|\ln\hat{p}_r(X_i) - \ln\hat{p}_N(X_i)\Big|. \tag{6.9}$$

While this simplifying assumption reduces the amount of computation required dramatically, it is still not enough to allow optimization of the prototype positions in a reasonable amount of time. Fukunaga & Hayes (1989) therefore uses the further constraint that the reduced set of prototypes must be a subset of the original data. A similar formulation is used with the nearest neighbour estimator in (Fukunaga & Mantock 1984).

To minimize J, the original data is divided into two sets, prototypes and non-prototypes, and pairs of points are switched between the two sets if the switch reduces J. A close examination of the method shows that the use of the log-ratio in the distortion measure is problematic, weighting relative errors at the expense of absolute errors. Regions of the space that could be well approximated as having zero density may require a prototype in the neighbourhood, in order that the reduced density is not orders of magnitude smaller than the original estimate, because that makes the log-ratio large even if the absolute difference is infinitesimal.

Xie, Laszlo & Ward (1993) looked at applying LBG to the DRDE problem. They noted that, for a particular type of estimator, Gersho (1979) had shown that when the number of prototypes r is large enough, the density function implied by the prototypes closely approximates the actual underlying density function from which the original data points were sampled. (Of course, because we are interested in r as small as possible, this analysis is questionable. In fact, Xie et al. (1993) present results where r is as small as 1.)

Their algorithm for classification simply uses the LBG clustering algorithm to reduce the number of prototypes in each class individually. Then the density for any class at a given point is

just the Parzen Windows (or kNN) estimate based on the reduced prototype set (Equation 6.7). The main problem with this method is the same as with all of the standard clustering techniques, which is that no explicit attention is paid to the underlying densities. The minimization is over the distances between points and prototypes, and the prototypes tend to spread out so that no point is too far from a prototype. The density generated by the reduced set is thus distorted from the original, full data estimate.

More recently, Babich & Camps (1996) introduced an algorithm they call Weighted Parzen Windows (WPW). This method uses a greedy clustering technique known as "agglomerative hierarchical clustering" in which, at each step, the closest two prototypes are merged. The crucial difference between this and previous methods is an extra *weight* parameter for each prototype, which is determined as the sum of the weights of the two merged prototypes, each original data point starting out with a weight of unity. The new center's location is the weighted sum of the coordinates of the two original prototypes. The difference between original and reduced density estimates, summed over the initial data point locations, is used as an error criterion. Merging ceases when either the error or the slope of the error becomes too large, and this determines the final number of prototypes.

This method compares well with the full Parzen Windows estimates in the classification experiments they present, giving only slight increases in leave-one-out error rates for fairly large reductions in the numbers of prototypes.

6.3 Weighted Vector Quantization

The WVQ algorithm was originally motivated by the work of Xie et al. (1993). However, our method also uses the insight found in (Babich & Camps 1996), that adding a weighting parameter to the prototypes can lead to more accurate density estimates.

We begin with the assumption in (Xie et al. 1993), that the VQ algorithm distributes the reduced prototype set in such a way that the resulting density is a good approximation to the

underlying density. This is clearly only true if the number of prototypes is large enough with respect to the number of modes in the distribution, and depends on how fast the distribution may be changing.

A simple example illustrates how VQ can fail when the number of prototypes is too small. Suppose the true distribution consists of two identical, well-separated Gaussians. Two well-placed prototypes could favorably represent such a distribution. Suppose instead that VQ is run with four prototypes. Due to the random initialization procedure, three of the four could end up within the domain of one of the Gaussians, and the resulting 3:1 weighting would mean that the true density was very poorly modelled.

While this is an extreme example, to a lesser extent it may happen quite often when the number of prototypes is not large enough. Because minimizing space is an issue, this domain may be an important one. Interestingly, if weighting of prototypes is allowed in the above example, the results are far better. A single prototype will be associated with all exemplars from one Gaussian, while the other three prototypes will divide the exemplars of the other Gaussian between them. Although the 3-prototype hump will be somewhat broader than that of the single prototype, *both will have roughly the same height*, instead of the 3:1 ratio of VQ.

The steps of the WVQ algorithm are given in Figure 6.1. Each class c_i is considered separately. This approach, like the previous two, is simple and efficient, and provides good results. Unlike (Fukunaga & Hayes 1989), prototypes are not restricted to the positions of the initial data points. Furthermore, the single integer weighting parameter adds relatively little to the amount of storage for each prototype (a smaller fraction in higher-dimensional spaces), but this simple addition potentially allows a small number of prototypes to provide as good an estimate as a non-weighted method using a much larger number.

WVQ can be considered as a kind of smooth histogram, with adaptive bin positions. This is similar to WPW, but whereas WPW is a greedy algorithm, always merging the closest prototypes, WVQ considers more interactions between the prototypes and the original data points while determining associations through the iterative VQ process. Another difference is

WVQ Algorithm:

STEP 1: For class c_i, with data points $\{X_1^i, ..., X_{N_i}^i\}$, choose the number of prototypes $r_i \leq N_i$. Initialize the prototype set to a random subset of r_i of the original data points.

STEP 2: Apply the LBG VQ algorithm to determine the reduced set of prototypes $\{Y_1^i, ..., Y_{r_i}^i\}$.

STEP 3: For each prototype $Y_j^i (j = 1...r_i)$, compute w_j^i, the weight of the prototype, according to the number of data points which are closer to Y_j^i than to any other prototype.

STEP 4: The Parzen density estimate of a point X is given by

$$\hat{P}_i(c_i|X) = \sum_{j=1}^{r_i} \frac{w_j^i}{N_i|H_i|} K[H^{-1}(X - Y_j^i)].$$

If the estimate is to be used to generate hypothesis rankings, we are done.

STEP 5: If the estimate is to be used for classification: for a given test vector X, the estimates $\hat{P}_i(c_i|X)$ are computed for each class c_i, and X is classified according to the class with the largest response.

Figure 6.1: Weighted Vector Quantization algorithm.

that WPW is deterministic, in that there is only one path between the unmerged set of points and the fully merged "set", which is a single prototype located at the centroid of the data. In contrast, the initial prototype set for WVQ is a randomly chosen subset of the original data, leading to minor variations in performance. Normally this will be an advantage, since the algorithm can be run several times with different initializations, and the solution that provides the lowest classification error on the training set can be kept.

A final difference between WPW and WVQ is that WPW uses the difference between the original and reduced set density estimates (summed over the original prototype positions) as a distortion measure to determine when merging should cease. WVQ assumes that using more prototypes gives more accurate estimates, and therefore advocates reducing the data only as much as necessary (e.g. until it can be fit into the machine's memory, or until the lookup time with the reduced number of points is acceptable).

The only difference between the VQ algorithm of Xie et al. (1993) and WVQ is the weight parameter of WVQ.

We next present some experiments to demonstrate the performance of WVQ, and to compare it specifically with VQ and WPW. The Parzen kernel was taken to be a Gaussian. The classification experiments were chosen to allow direct comparison with previous approaches, and in these the covariance matrix H was set to match the one used in the reference from which the experiment was taken. WVQ was also tested using samples from the model database, to demonstrate that it also works on distributions important for the indexing problem. There, H had the simpler form of a diagonal matrix, and the dimensionality weights were determined as in Section 3.6.3, either set manually or using the VSM algorithm.

6.3.1 Classification Experiments

The first two experiments in this section use synthetic probability distributions where ground truth is available. The Bayes' error in each case provides a basis for judging WVQ performance. The first experiment, with 2D Gaussians, shows that the method works for simple distributions.

The second experiment, Gaussians in an 8D space, is more difficult because the sample sets are relatively sparse for the dimensionality of the space. The final classification experiment shows that WVQ has good performance on a well-known set of real data (Fisher's IRIS data (Fisher 1936)).

2D Gaussians

In this experiment (from (Babich & Camps 1996), p. 569), there are two classes composed of Gaussians. One class is bimodal, and the other is unimodal and rests in the "valley" of the bimodal distribution. The densities of the classes are $p(\vec{\mathbf{x}}|c_1) = N(\mu_1, I)$ and $p(\vec{\mathbf{x}}|c_2) = 0.5N(\mu_2, I) + 0.5N(\mu_3, I)$. The Gaussian centers are at $\mu_1 = [2.5, 2.5]$, $\mu_2 = [0, 0]$, and $\mu_3 = [5, 5]$, and the covariance matrices are equal to the identity matrix I. The window H for the Parzen kernel was also taken to be I, as in (Babich & Camps 1996). The Bayes error for this system was determined to be 5.5% using the known distributions to classify 10,000 random samples, 5000 from each class.

100 random samples were generated for each class, and the WVQ algorithm applied for each of several values of N_r, the reduced number of prototypes. (Here N_r is the total for both classes.) The classification error was determined using the leave-one-out error measure. In this approach, WVQ is applied once for each of the original data points (in this case 200 times). Each time, a single point is removed from the original data and set aside, and a reduced set of prototypes is determined without the influence of the removed point. That point is then tested against the resulting classifier to see if it is properly classified. (The estimator used in this, and all of the "Classification experiments" was Parzen Windows.) The process is repeated for each original point, and the compiled results over all 200 points determine the percentage error.

50 runs with different initial sample sets were acquired, and the data is plotted in Figure 6.2. The large jump in LV error rate when the number of prototypes per class is equal to one is because a single prototype is being used to represent a bimodal distribution. WVQ performance compares favorably with Figure 2c of (Babich & Camps 1996), especially for

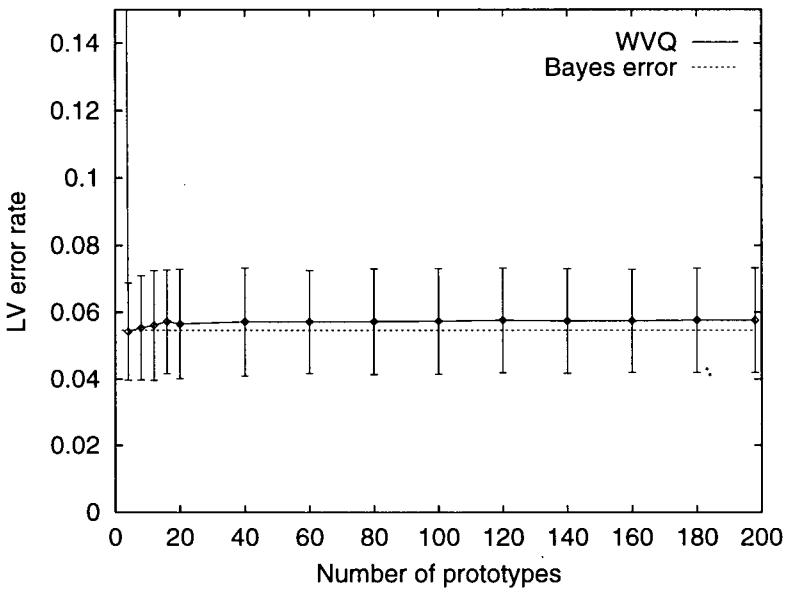


Figure 6.2: 2D Gaussian results.

smaller numbers of prototypes. The average error with our method actually decreases slightly (down to $N_r = 2$), due to the simple form of the distribution. Where it does better than WPW the reason is that it is less greedy, and does some optimization of prototype positions using the iterative VQ algorithm.

$8D$ Gaussians

This experiment originates in (Fukunaga 1990), p.556, and has also been used to test the VQ ((Xie et al. 1993), p.1328) and WPW ((Babich & Camps 1996), p.569) algorithms. There are two classes composed of $8D$ Gaussians, both of which are bimodal and offset from one another. The densities of the classes are $p(\vec{\mathbf{x}}|c_1) = 0.5N(\mu_1, I) + 0.5N(\mu_2, I)$ and $p(\vec{\mathbf{x}}|c_2) = 0.5N(\mu_3, I) + 0.5N(\mu_4, I)$, with Gaussian centers at $\mu_1 = [0\ 0 \ldots 0]$, $\mu_2 = [6.58\ 0 \ldots 0]$, $\mu_3 = [3.29\ 0 \ldots 0]$, and $\mu_4 = [9.87\ 0 \ldots 0]$. The covariance matrices are equal to the identity matrix I, and the window H was taken to be $1.5^2 \times I$ as in (Xie et al. 1993). The Bayes error for this system is 7.5%.

The size of the original data set was 150 samples per class, and the classification error

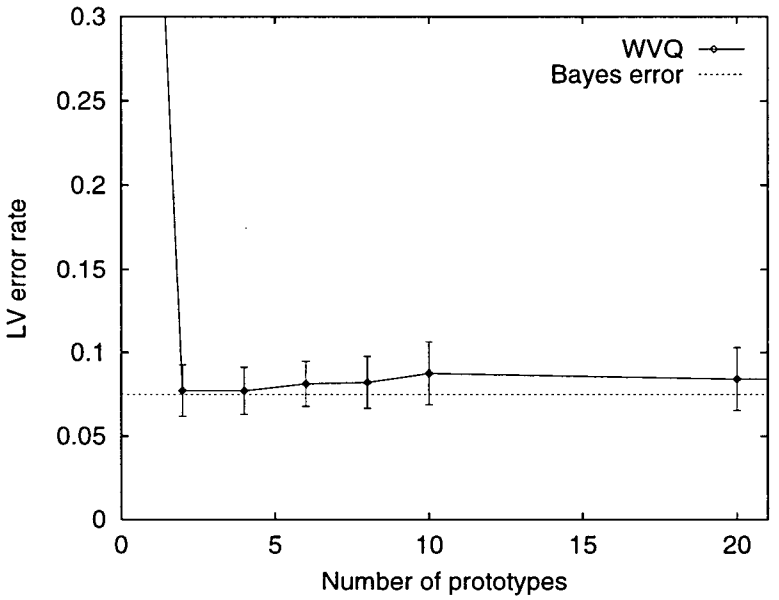


Figure 6.3: 8D Gaussian results.

was measured using the leave-one-out procedure. Results are presented in Figure 6.3, averaged over 50 runs with different sample sets. The large jump in LV error rate when the number of prototypes per class is equal to one is because a single prototypes are being used to represent bimodal distributions. These results are comparable to those of Xie et al. (1993) and Babich & Camps (1996), and are an improvement over Fukunaga's reduced Parzen classifier (Fukunaga & Hayes 1989). This reinforces the idea that WVQ is able to represent simple distributions using a small number of prototypes.

Iris Data

This experiment uses the classic data set of Fisher (1936), which contains 50 samples of each of three classes of Iris, characterized by 4 real-valued attributes. One class is linearly separable from the other two, but the other two are not linearly separable from each other. The window H is taken to be $1.30\hat{\Sigma}^{0.5}$, as in (Babich & Camps 1996), where $\hat{\Sigma}^{0.5}$ is the symmetric square root of the empirical covariance matrix.

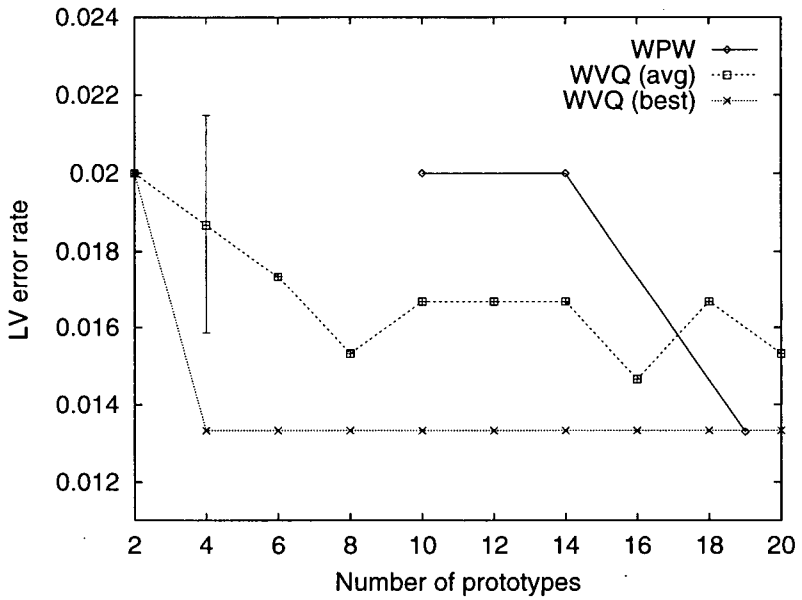


Figure 6.4: Iris results.

Results of applying the WVQ algorithm are presented in Figure 6.4. The algorithm was run 10 times with different initial positions for the prototypes. The error rates for the average and best runs are shown, along with the WPW results from (Babich & Camps 1996). The WPW algorithm is deterministic, so there are no errorbars associated with the data. For the WVQ "average" data, the errorbars are relatively large and interfere with the interpretation of the plots, so they are omitted. (An indicative example is drawn for $N_r = 4$.) The WVQ "best" data for each value of N_r is the lowest leave-one-out error rate over all 10 runs.

The "best" run shows that WVQ is able to maintain a lower error rate down to a smaller number of prototypes, 4 per class versus at best 15 for WPW. In only a single case out of all runs for all values of N_r did WVQ give a higher error rate than WPW. This implies that if there is only time for one run (unlikely), then WVQ will almost never do worse than WPW, and on average appears to do better. If time permits, multiple runs of WVQ can be made and the one providing the lowest error rate chosen, and that should also be at least as good as WPW.

6.3.2 Incremental Learning

We now present experiments using the model database. These will demonstrate both an improvement in recognition rate due to incremental learning alone, and that WVQ can be used in conjunction with incremental learning to improve performance without increasing the memory requirements of the system. In all of the following experiments, procedures were similar to those in experimental Section 4.6: the WkNN estimator was used; the number of nearest neighbours recovered was 10; BBF search was used, and the maximum number of leaf nodes examined on any query was 100; all synthetic test images and update images had 1.5 pixel Gaussian noise added to segment endpoints; and 4-segment chains were used as the grouping type, which provided a 6D index space. The "infinite memory" updates referred to below are those for which the learning data is added to the index with no clustering, so the storage requirements increase. This contrasts with WVQ, in which the number of prototypes is reduced back to the original, pre-update level.

In the WVQ algorithm, the first step is to choose the size of the reduced prototype set for each class. In the indexing problem, there are a great many classes (the underlying 3D groupings), each of which has its own number of initial points from the training stage and update points from the incremental learning stage. In order to demonstrate improved performance at no increase in memory, it is necessary to cluster the full training and update sets back to the original size (N_{init}) of the full training set.

One way to do this is to treat each class independently, reducing each one to its original number of prototypes. However, this limits the effectiveness of incremental learning, since the accuracy of a probability estimate is determined in large part by the number of prototypes available to represent it, and this number would be fixed for each grouping during training.

A more equitable solution is to pool together all training and update samples from all groupings, and to randomly choose N_{init} prototypes from this set. The likelihood is that each grouping will be allotted a number of prototypes approximately equal to its *a priori* probability

of appearance in the combined training plus update set. Repeating this process over many updates will eventually overwhelm the synthetic training data with real image data, optimizing the performance of the indexing system.

Experiment 1

In this experiment, we look at the improvement in performance when incremental learning is applied to one of the models in the database (call it the *test object*) but not to the others. This corresponds to the situation when one object is more likely to appear in images than others. Index updates in this scenario will both increase the *a priori* probability of the test object relative to other objects, and fill in some of the gaps that may exist in the original set of training views of the test object.

The experimental procedure was as follows. In the training stage, 10 random views of each of the 10 database objects were generated, and 4-segment chain groupings extracted and stored in the index. Recognition performance was judged using a test set of 100 random images of the test object. This same set was used to test performance after the learning update.

To apply incremental learning, a set of 50 random images of the test object was used. For each of these, the recognition algorithm was run and, when successful, matched image groupings were saved. Once all of the update images had been run, the saved groupings were added to the index. The 100 image set was used to test performance in this "infinite memory" scenario, then the WVQ algorithm was applied to reduce the number of prototypes back to the original, pre-update level and performance was tested again. Note that when NN search was applied after the WVQ update, the effective number of neighbours recovered increased (from the original value of $k = 10$) because some prototypes had weights attached. This was interpreted to be the increase of k with increasing numbers of stored points suggested by Duda & Hart (1973) (and others). For more extensive or ongoing updates a more careful program to increase k would be required.

The entire process was applied 10 times, with each of the database objects standing

	Infinite		WVQ	
	before	after	before	after
% Found	63.1	74.7	63.1	73.7
Rank	59	27	59	31

Table 6.1: Results for Experiment 1.

as the test object. The average number of original points was 8800 from 4300 groupings, and the average number of update points was 1200 from 450 groupings. Results are presented in Table 6.1. The percentage of objects found increases and the average rank decreases by roughly the same amount for each update method. The application of WVQ to reduce the number of prototypes slightly reduced the positive effects, but by only a small fraction of the overall improvement. Therefore, we can say that there has been an improvement in performance with essentially no increase in memory (just the amount required to store the prototype weights).

Experiment 2

In this experiment, synthetic images are again used to provide data for incremental learning, but unlike the previous experiment, *a priori* probabilities for all models are kept similar. This allows us to investigate what happens when particular groupings are more likely to appear in images than are other groupings. One example of this scenario would be if there were some constraint on the pose of an object with respect to the camera, perhaps if the camera were fixed in position and the object has only a small number of resting positions. This example was used in the following experiment, but the results apply to any situation in which the probability of observing some features is increased relative to others.

The training stage consisted of 25 random images for each of the 10 models in the database. Synthetic test images of one database objects (the *test object*) were generated, with the constraint that the inclination of the camera with respect to the object was fixed at $\psi = 55°$ above the horizontal (Figure 6.5). 15 images at equally spaced increments of $\theta = 24°$ were generated, and the recognition performance tested. This "constrained test set" was also used

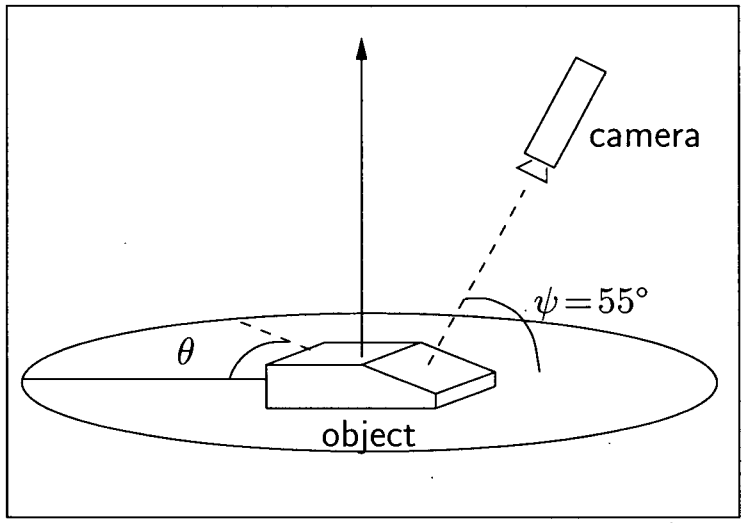


Figure 6.5: Experimental setup for experiment 2.

to test performance after the learning update.

Incremental learning was applied with two separate types of update image. For the test object, 15 additional constrained images were used, at the same fixed inclination, but offset from the test set by $\theta = 12°$. For the rest of the objects, images from random viewpoints were used. To make sure that the *a priori* probabilities for the non-test objects remained at least as great as those for the test object, images were generated until 15 successful recognitions of each non-test object had been achieved. After adding the new data to the index, the earlier "constrained test set" was used to gauge performance in the "infinite memory" regime. Then the WVQ algorithm was applied to reduce the number of prototypes back to the pre-update level and performance was tested again. The number of original points was 21000 from 6400 groupings, and the number of update points was 5400 from 3100 groupings.

Results are presented in Table 6.2. As in the previous experiment, we can observe an improvement in performance with no increase in memory when WVQ is used. Both update methods push the number of successful recognitions to 14 out of 15, although in doing so WVQ raises the average number of verifications required from 43 up to 93.

	Infinite		WVQ	
	before	after	before	after
# found	11	14	11	14
Rank	43	29	43	93

Table 6.2: Results for Experiment 2.

Experiment 3

In this experiment, incremental learning was applied using real images. Again, *a priori* probabilities for all models were kept similar. The training stage consisted of 10 images of each object from random viewpoints in a hemisphere. The test and update images were from a set of 208 views of the noteholder object (the test object). These were 4 sets of 52 well-spaced views of the upper hemisphere of the object, each set taken under different lighting conditions. (Some of these images are shown in Figure 4.18.)

The recognition performance was initially tested using 25 of these images, and the same set was used to test performance after the learning update. The remaining 187 images were used for incremental learning. First of all, indexing and recognition were applied to these images, and when recognition was successful, matched image groupings were set aside for later inclusion in the index. To keep *a priori* probabilities similar for all models, after the 187 image update set had been run, recognition was applied to random synthetic views of the remaining objects until each object had accumulated at least as many update vectors as the test object. After adding the new data to the index, the original set of 25 images was used to test performance in the "infinite memory" regime. Then the WVQ algorithm was applied to reduce the number of prototypes back to the pre-update level, and performance was tested again. The number of original points was 9800 from 4300 groupings, and the number of update points was 1300 for each model (a total of 13000).

Results are presented in Table 6.3. Again, both update methods increase the number of successful recognitions, and in this case each method also reduces the average number of

	Infinite		WVQ	
	before	after	before	after
# found	9	15	9	12
Rank	24	5	24	19

Table 6.3: Results for Experiment 3.

verifications required for recognition.

6.3.3 Summary

The experiments in this chapter have demonstrated that it is possible to improve performance using incremental learning without increasing memory requirements, by the use of a clustering technique. They also indicate that incremental learning is able to "extend the range" of recognition, by filling in some of the gaps between training views. In a similar way, given a full set of orthographic views of an object, the range of recognition can be extended to include stronger and stronger perspective views.

One way that memory might be further optimized would be if it were possible to characterize some of the distributions of certain types of grouping. For example, if it was known that planar segment chains of a given size could have no more than a fixed number of modes in their distributions, it might be possible to limit the number of prototypes used to represent them. This type of analysis has not yet been done, and is left as an open question.

Chapter 7

Semi-Automated Model Building Using Structure-From-Motion

7.1 Introduction

One of the practical details that has frequently been glossed over in Model-based Object Recognition is how to acquire the models in the first place. To date, model building has been a completely manual undertaking, in which a user makes direct measurements of the 3D dimensions of the objects by hand. Ideally, this process would be automated such that only a set of 2D images of the object would be required. These would form the input to a structure-from-motion (SFM) algorithm that would solve for the camera viewpoints of the images relative to the object, and then reconstruct the 3D positions of certain features which would comprise the model. Unfortunately, the state of the art in SFM is not yet good enough for this process to be fully automated. Feature tracking between images is not 100% effective and, more importantly, higher-level reasoning is required to determine which features are relevant for inclusion into the model.

One approach that has potential for the near future is to use a *partially* automated process. In this scenario, human intervention is required at specified points in the procedure,

e.g. to select desirable features from the images, but the difficult task of 3D reconstruction is accomplished automatically. As a proof of concept, we adopt a recent SFM algorithm (Taylor & Kriegman 1995), which has the distinct advantage that a C-code implementation for the method is available by *ftp*. The algorithm uses a set of 2D image edges to reconstruct 3D straight edges for a model.

The reconstructed segments provide the basis for a 3D wire-frame model of the imaged object. However, because each 3D segment is "independently" reconstructed, i.e., with no explicit constraints relating segments to surfaces or to other segments, edges that should lie on the same object face may not, due to noise in the image. Furthermore, the algorithm does not provide accurate endpoint localization. By adding a post-processing phase which allows specification of constraints by a user, segments can be forced to coterminate and/or to lie on a planar face. This stage cleans up the SFM output, and endows the models with face information that the original reconstructed segments lacked. This is important if the model is to be used for recognition, so that self-occluded segments can be removed whenever the model is projected. In the following, we describe the steps required to implement the model building procedure. For more details, including pointers to the implementated modules, see (Beis 1996).

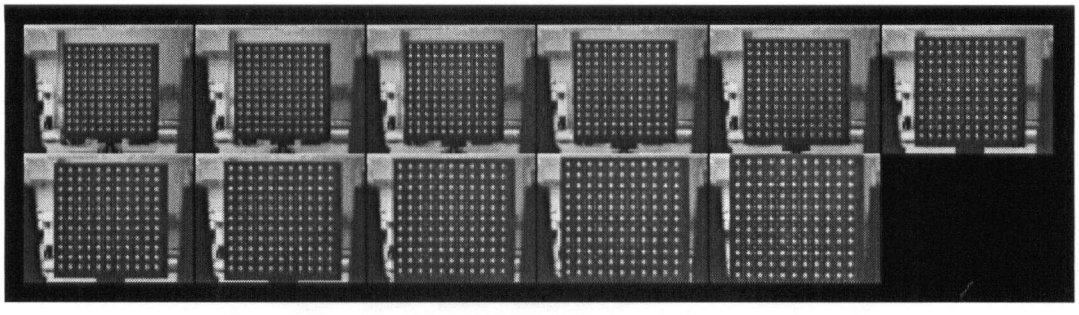


Figure 7.1: 11x11x11 cube of points used for camera calibration.

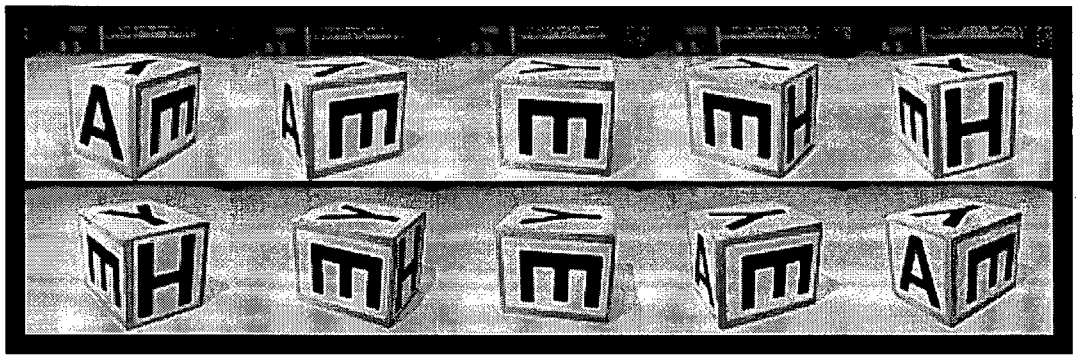


Figure 7.2: Ten images of the object to be modeled.

7.2 Camera Calibration

The camera calibration algorithm is due to Tsai (1987). While a simple version of this algorithm uses only a single image containing a planar grid of points, the more accurate version requires imaging a cubic grid of points in space. Figure 7.1 shows the 11 images that were used for calibration. The RMS errors for the alignment of actual with modelled points were 0.3 pixels in x and 0.5 pixels in y. The maximum errors were 1.1 pixels in x and 1.4 pixels in y.

7.3 Take Pictures

In this step, pictures of the object are taken (Figure 7.2). One of the images is to be considered the *base* image, and estimates of the camera rotational offsets for each of the other images must be recorded. Note that these only have to be rough estimates, but the better the estimates, the faster the SFM algorithm is likely to converge.

Thus, while the algorithm is designed to be general-purpose, its use in model-building should be more successful than for, e.g. piloting a robot, because the environment can be controlled during image acquisition. It is straight-forward to obtain very good estimates of the camera positions, and the light sources can be engineered to highlight the desired features (edges) in each view. Another positive aspect of model building as an application of SFM is

that the processing time is not insignificant. Whereas a real-time application may not be able to wait some minutes for the computation to be completed, this is not an issue for building models.

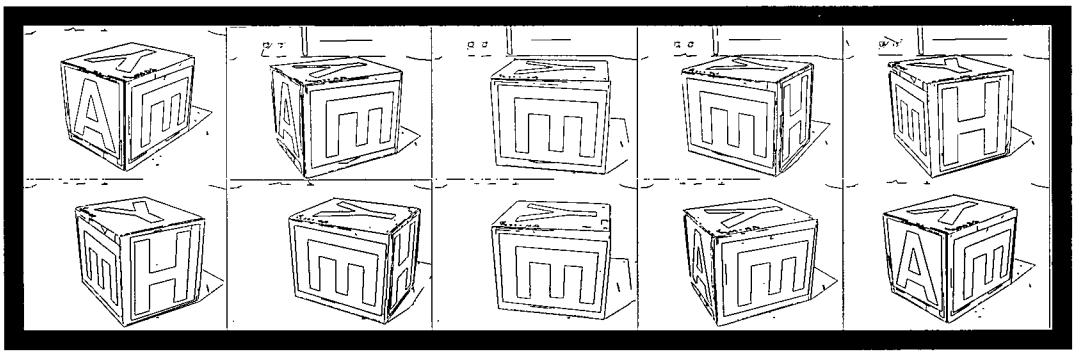


Figure 7.3: Edges extracted from image set (σ of Gaussian kernel in the Laplacian-of-Gaussian edge detector was 0.5 pixels).

7.4 Extract Edges

The images must be processed to extract edges, link the edgels, and finally, segment the linked chains of edgels into a set of straight segments. The various parameters of the operations should be tuned to give accurate localization (e.g., small window size for the edge detection kernel). As well, the thresholds should be set to recover more edges rather than fewer, because the user will later select only those edges desired for the model. The edges recovered for the set of ten example images are shown in Figure 7.3.

7.5 Label Edges

Next the edges in each of the images must be labelled. This serves two purposes, both to select which features are to be included in the model, and to indicate the correspondences between edges in different images which are required by the SFM algorithm. Note that the current SFM algorithm can deal with broken edges within an image, so all significant portions of an edge are

tagged with the same label. This is the first stage where extensive user interaction is necessary, and a simple program has been developed to facilitate the process. Results from this stage are shown in Figure 7.4.

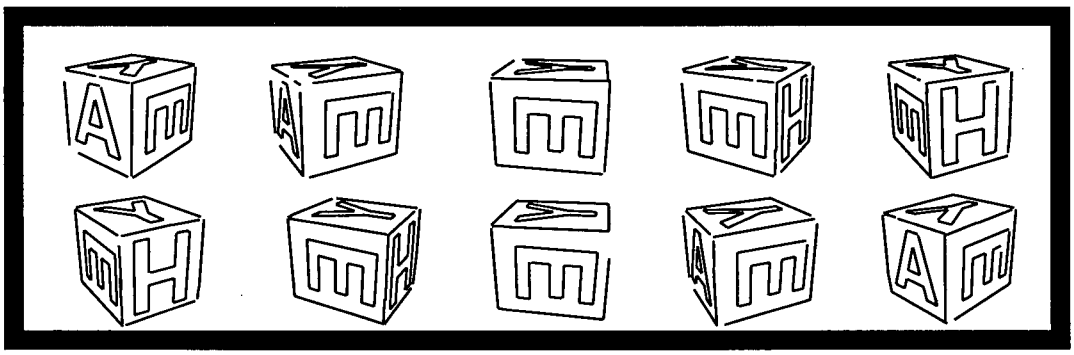


Figure 7.4: Edges the user has chosen to include in the model, for all ten images of the object.

For each labelled edge set, the intrinsic camera parameters determined in the calibration stage are used to transform the segment endpoints from pixels to mm, in particular to remove any aspect ratio or radial distortion transformations. Taylor and Kriegman further normalize the data by dividing by the camera focal length, and the optimization takes place in this unitless, normalized domain. The latter step is only necessary to simplify explanation of the technique, as there will always be a global scale factor that can only be determined by direct measurement.

7.6 SFM Algorithm

The SFM algorithm proceeds in stages, first using simple constraints to obtain estimates for some of the parameters (e.g. the 3D line orientations only, without their translations), proceeding in later stages to solve for more and more of the unknowns. The full set of unknowns includes the parameters for the 3D line equations taken with respect to the base frame, and the extrinsic parameters for the other camera locations (rotations and translations), with respect

to the same base frame.

Let $\vec{\mathbf{q}}_j$ be the extrinsic parameters of the j^{th} camera (the 6 parameters for position and orientation). Let $\vec{\mathbf{p}}_i$ be the parameters defining the i^{th} 3D model line. Let $\vec{\mathbf{u}}_{ij}$ be the measurement of the projection of feature i in image j. Finally, let *Error* be a positive, real-valued function that measures the difference between the observed image feature $\vec{\mathbf{u}}_{ij}$ and the expected image feature $F(\vec{\mathbf{p}}_i, \vec{\mathbf{q}}_j)$, where F is a function that defines the image formation process. Then the goal of the SFM algorithm is to minimize the objective function

$$O = \sum_{j=1}^{m} \sum_{i=1}^{n} Error(F(\vec{\mathbf{p}}_i, \vec{\mathbf{q}}_j), \vec{\mathbf{u}}_{ij}) \tag{7.1}$$

To understand the algorithm, it is useful to note that a 3D line can be represented as a unit vector $\hat{\mathbf{v}}$ giving the orientation of the line, and a unique vector $\vec{\mathbf{d}}$ that originates at the origin and is perpendicular to $\hat{\mathbf{v}}$ (see Figure 7.5). Each 3D line requires 4 parameters to specify, three for the vector $\vec{\mathbf{d}}$, and one more for the rotation of $\hat{\mathbf{v}}$ about $\vec{\mathbf{d}}$. (Note that the optimization solves for the parameters of the line, and only afterwards tries to reconstruct the segment endpoints.) The error function *Error* is the integral of the squared distance between the image segment and the projection of the estimated 3D line, over the length of the image segment. This evaluates in closed form to the simple expression $E = \frac{l}{3}(h_1^2 + h_1 h_2 + h_2^2)$, where h_1 and h_2 are the perpendicular distances of the segment endpoints from the projected 3D line, and l is the length of the segment. This measure is simple, easy to compute, and proportional to l so that longer segments, which in general are more accurately localized, are given greater weighting.

The extensive optimization is broken into stages. The first division is into two regimes, a global and a local optimization. Because the objective function is non-linear, we have to worry about local minima. Therefore, to try to ensure a good solution, the global optimization works by starting off the local stage at several different sets of initial parameter estimates. The ranges over which these parameters are initialized depend on uncertainties specified by the user (e.g. camera orientations known to $\pm 20°$). If at some point the local optimization produces a solution with a small enough error, the global re-initialization ceases. As mentioned earlier,

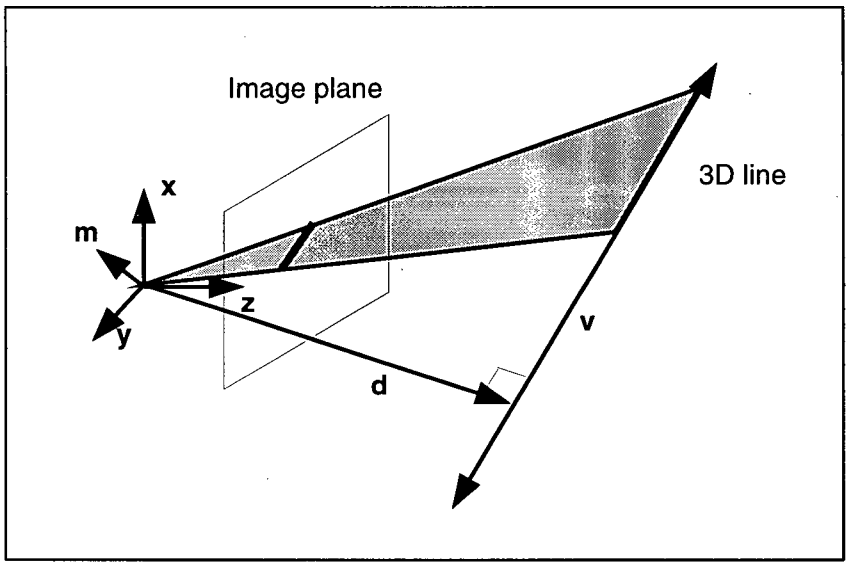


Figure 7.5: Camera geometry and definition of 3D lines within the coordinate system associated with each image.

when building models camera orientation estimates can be known with good accuracy, so the global stage should never require more than a single iteration.

The local optimization is itself broken into 4 stages. The key here is that with some manipulation, various parameters can be isolated and solved for, while the other parameters are kept fixed. For this to work, the fixed parameters (e.g. camera rotations) must begin with values in the neighbourhood of a solution, and this is why the global stage is necessary.

7.6.1 Stage A

One of the images is first chosen to be the base frame. The 3D lines and remaining camera positions/orientations will be determined relative to this frame. Let this frame be indexed by w (world), and any other frame be indexed by c (camera). Within any of the camera frames, by the definition of $\vec{\mathbf{m}}$ (see Figure 7.5) we have

$$^{c}\vec{\mathbf{m}} \perp {}^{c}\vec{\mathbf{v}} \tag{7.2}$$

or

$$ {}^c\vec{\mathbf{m}} \cdot {}^c\vec{\mathbf{v}} = 0 \tag{7.3} $$

$$ {}^c\vec{\mathbf{m}}({}^c_wR\,{}^w\vec{\mathbf{v}}) = 0. \tag{7.4} $$

Here ${}^c_wR$ is the rotation matrix to transform from the world to the camera frame. This relation only holds for perfect data, but it can be used to measure how well any line orientation $\vec{\mathbf{v}}$ fits a given camera orientation ${}^c_wR$. Note that $\vec{\mathbf{m}}$ for each camera can be determined by recalling that the focal length is exactly 1 due to the initial normalizations, so that

$$ {}^c\vec{\mathbf{m}} = ({}^cx_1, {}^cy_1, 1) \wedge ({}^cx_2, {}^cy_2, 1), \tag{7.5} $$

where these vectors point from the camera frame origin to the segment endpoints in the image. These are fixed by the camera calibration and image data, not to be solved for in th SFM optimization. A useful "partial" objective function can be created based on Equation 7.4:

$$ C_1 = \sum_{j=1}^{m}\sum_{i=1}^{n}(\hat{\mathbf{m}}_{ij}R_j\vec{\mathbf{v}}_i)^2. \tag{7.6} $$

Here, i indexes segments and j indexes the camera frame for each image. Each segment in general appears in some but not all images. With perfect data the minimum of this function will be zero. Otherwise, it measures how well 3D line segment *orientations* match with the image data, based on the initial estimates of camera orientation R_j. Note that C_1 is *separable* by line segment:

$$ C_1 = \sum_{i=1}^{n} C_{A_i} \tag{7.7} $$

where

$$ C_{A_i} = \sum_{j=1}^{m}(\hat{\mathbf{m}}_{ij}R_j\vec{\mathbf{v}}_i)^2. \tag{7.8} $$

With R_j fixed, it is possible to solve for the orientation $\vec{\mathbf{v}}_i$ of each 3D line segment *independently*, using singular value decomposition. Recall that the R_j are estimates supplied by the global stage of the overall optimization. If these are not close enough to the true values, the solutions for the line orientations will be incorrect, and the remainder of the optimization will not recover a good solution (it will find a poor local minimum of the objective function, Equation 7.6).

7.6.2 Stage B

In this stage, the R_j parameters are allowed to vary along with the v, and the same partial objective function is solved again. The method of solution is not indicated in the paper.

7.6.3 Stage C

This stage uses a different partial objective function to arrive at estimates of the translational parameters for each camera (denoted by $\vec{\mathbf{t}}_j$) and for the 3D lines $\vec{\mathbf{d}}_i$ (see Figure 7.5). A new constraint is used, which is that each 3D line in a camera frame c satisfies

$$^c\vec{\mathbf{m}} \perp ^c \vec{\mathbf{d}} \tag{7.9}$$

or

$$^c\vec{\mathbf{m}} \cdot ^c \vec{\mathbf{d}} = 0 \tag{7.10}$$

$$^c\vec{\mathbf{m}}(^c_w R(^w\vec{\mathbf{d}} - ^c_w \vec{\mathbf{t}})) = 0. \tag{7.11}$$

Again, this is true only for perfect data, where $(^w\vec{\mathbf{d}} - ^c_w\vec{\mathbf{t}})$ is the expected value of the world (base camera) $\vec{\mathbf{d}}$ vector for a given segment, after translating and rotating it into camera frame c. The following partial objective function can measure how well our data fits this requirement:

$$C_2 = \sum_{j=1}^{m}\sum_{i=1}^{n}(\hat{\mathbf{m}}_{ij} R_j(\vec{\mathbf{d}}_i - \vec{\mathbf{t}}_j))^2. \tag{7.12}$$

This can be massaged into a "simple quadratic form" in 5 parameters which specify $\vec{\mathbf{d}}_i$ and $\vec{\mathbf{t}}_j$. (Of the six degrees of freedom in each $\vec{\mathbf{d}}_i$, $\vec{\mathbf{t}}_j$ pair, one has already been eliminated since each of the $\vec{\mathbf{d}}_i$ must be perpendicular to its corresponding $\vec{\mathbf{v}}_i$ as estimated by the previous stage.) The R_j parameters are fixed as in Stage A, and C_2 can then be minimized by linear least squares providing estimates for the $\vec{\mathbf{d}}_i$ and $\vec{\mathbf{t}}_j$.

7.6.4 Stage D

This is the final stage, which is a large, non-linear optimization over all parameters, using the initial estimates from the previous stages. The algorithm is similar to that of Smith (1993).

The total number of parameters is given by

$$N = 4L + 3(C - 1) - 1 \tag{7.13}$$

where L is the number of lines in the model and C is the number of camera views. In our simple box example, with roughly 40 lines and 10 views, there are already 186 parameters to determine! It is therefore important for efficiency reasons to use stages A to C, so that stage D starts off fairly close to the final set of parameters. If the problem becomes too large, it should be possible to break it into separate subsets of images.

7.7 Postprocessing

If some of the model faces are known to be planar, the results of the reconstruction can be improved. The SFM algorithm does not allow relationships (constraints) to be specified between segments during the optimization. Nor does it guarantee accurate endpoint determination, since infinite 3D lines are first computed, then image endpoints are back-projected onto the lines to generate segments. A postprocessing stage is useful to (i) constrain a set of segments to lie on a planar face, and (ii) to force pairs, triples, etc., of segments to coterminate at corners, trihedral vertices, etc. Postprocessing consists of three steps, as follows.

7.7.1 Labelling of Segments

Firstly, all segments belonging to a given planar face must be labelled by the user. Secondly, segments which coterminate are also marked. A program which provides these functionalities has been implemented in which, to date, the most complex coterminations that can be handled are trihedral corners.

7.7.2 Determining Plane Equations

Given the assignment of segments to a face, a best-fit plane can be determined as that which minimizes the least-squares distance between the plane and the endpoints of all associated

segments.

7.7.3 Determining Segment Endpoint Locations

Once the planes have been computed, each segment is projected onto those planes of which it is a member. This is a single plane in the case of surface markings, or two planes if the segment is an edge at the intersection of two faces. After projection, final positions for the endpoints are determined by intersecting the appropriate lines and/or planes. For each endpoint, there are currently four possible cases for the segment it is part of:

(a) The segment is associated with one plane and no other segments (it is a "hanging" edge). Then we are done.

(b) The segment is associated with one plane and coterminates with one other segment (it is a surface marking). The endpoint is determined as the intersection of the two segments. Since both lie on the same plane, so will the point of intersection.

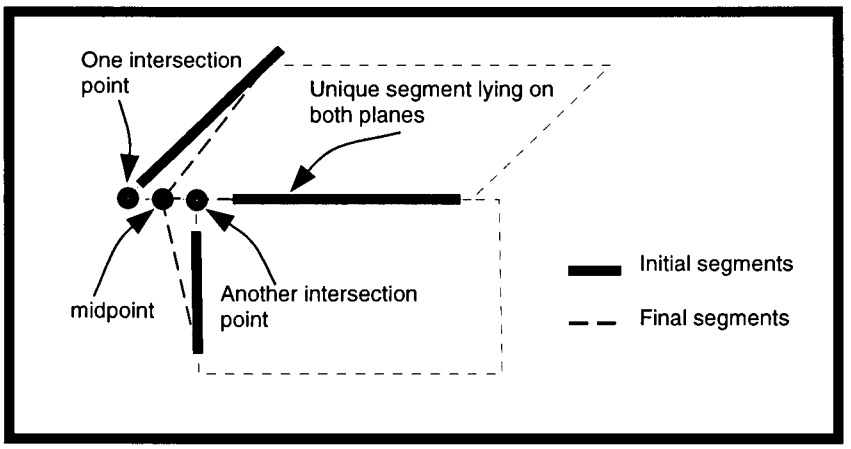


Figure 7.6: Determination of endpoint for trihedral corner when only two of the three plane equations are known.

(c) The segment is associated with two planes and coterminates with two other segments (it is a trihedral corner with one missing plane, the latter arising when some object face has

not been seen from enough views to admit reconstruction: see Figure 7.6). In this case, there will be a unique segment at the intersection of the two associated planes, while the two other segments will be associated with only one plane each. First, each of the two non-unique segments are intersected with the unique segment. Since the latter belongs to both planes, both intersection points will lie on the line of intersection of the two known planes. The midpoint of these intersection points is taken as the endpoint for all three segments.

(d) The segment is associated with three planes and coterminates with two other segments (it is a trihedral corner). The intersection of the three planes is taken as the endpoint for all three segments.

The improvements due to post-processing can be seen in Figure 7.7. It is somewhat difficult to tell that the segments in the "before" panel are not co-planar. In fact they are nearly so, and this is obviously a requirement for the model to be accurate, with or without the postprocessing stage. There are, however, several clear examples of corners where the segments do not coterminate before postprocessing.

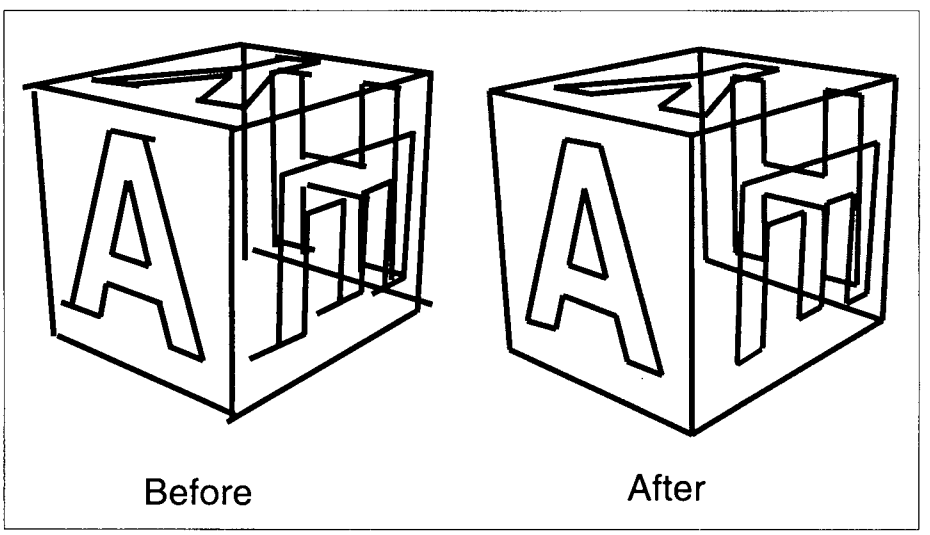


Figure 7.7: Results of SFM algorithm before (LHS) and after (RHS) the postprocessing stage.

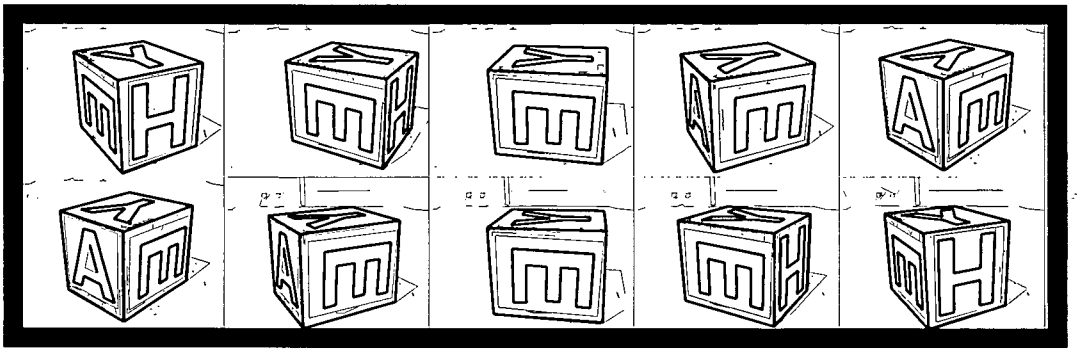


Figure 7.8: Examples of recognition using reconstructed model.

To demonstrate the accuracy of the reconstructed model, the recognition algorithm described in previous chapters was applied, using the original SFM images as test images and the reconstructed box as the model database. Indexing and recognition succeeded for all images, and Figure 7.8 shows the results with the model overlaid in bold on the edge-detected images. In each case, the model makes a tight fit to the image edges. As mentioned earlier, one useful consequence of having models with face information instead of just wire-frame models is that it becomes possible to do hidden-line removal. In these overlays, the back faces have been removed (unlike in Figure 7.7), which results in fewer matching errors during the alignment stage of recognition.

7.8 Summary

In this chapter, we have supplied an example of semi-automated model building. This is a proof-of-concept that with minimal user intervention, limited to edge labelling, object models useful for recognition can be reconstructed from a set of example images. In a separate test with images of a different object, the optimization procedure failed for unknown reasons. Thus, there is work to be done to make the procedure robust. However, because aspects of the problem can be well-constrained in this application of structure-from-motion (e.g. initial estimates for the

relative camera positions), it appears that there are no major technical difficulties to overcome before a good working system can be built.

Similarly, since motion can be constrained to be as slow as desired between images, feature tracking should also work very well. An even more automated version of this procedure would then be possible, in which the user would only need to label features in the first image, or when they first emerge from being self-occluded, and the tracking algorithm would then propagate the labels to the remaining images in a sequence.

Chapter 8

Conclusion

8.1 Overview of the Approach

In this dissertation we have investigated the problem of indexing a database of 3D object models using data available from single 2D grey-scale digital camera images. In formulating a solution, we have not assumed that invariant descriptors of shape are available. By using probability distributions of how complex local feature sets appear in images, the method is able to efficiently and accurately determine likely shape matches between an image and a model database.

The system consists of four main components, three of which comprise the index itself — the index key, the index data structure, and the stored data — plus an incremental learning procedure designed to improve indexing performance over time. In more detail:

Index key: The feature sets stored in and retrieved from the index were chosen to be specific types of non-accidental groupings of straight edge segments which exhibit the perceptually significant properties of either parallelism or cotermination. Experiments showed that, when detectable, groupings of only 3 or 4 segments were sufficient to discriminate between the models in a small database. Because the indexing technique is designed to work with abstract feature vectors, the set of groupings is extensible to include other feature types representing,

for example, curved edges, color, and texture. These additional features would be important for handling a wider variety of object classes and larger model databases.

Data structure: kd-trees were used for storing and retrieving data, in contrast with much previous work in model-based recognition where hash tables have been applied. In Chapter 5 we demonstrated that for difficult indexing problems, in higher-dimensional spaces and with larger model databases, the performance of hash tables breaks down. Also in that chapter, we presented an approximate nearest neighbour kd-tree search algorithm ("Best-Bin First" search). Experiments demonstrated that NN lookup remained efficient as feature complexity (dimension of index space) and number of models (number of stored points) increased. For example, in a 12D space with 300,000 stored points, BBF recovered on average 92% of the closest neighbours while examining at most 200 leaf nodes per query. These results indicate that the overall indexing approach scales well with model database size.

Stored data: What is effectively stored in the index are probability distributions of the 2D image appearance of 3D feature sets as they vary with viewpoint. This is the first indexing work that uses probabilities due to view variation to compute *a posteriori* probabilities that a given image shape corresponds to a specific object shape. A benefit of this approach is that a larger class of objects can be recognized than for those algorithms in which invariants are required. (Other methods that deal with probabilities use them to model the noise in "invariant" feature values, or else they deal with generic rather that specific object shapes.) Experiments in Chapter 4 demonstrated the utility of ranking hypotheses according to the estimated probability of correctness. Furthermore, although invariants were not used, the amount of memory required was not excessive: the 10 model database used less than 3MB.

Incremental learning: Our approach suggests incremental learning be applied throughout the working life of a recognition system, to continually update probabilities according to which features, and feature groupings, are currently being detected in real images. Because probability

updates involve adding new points to the index structure, an important consideration is that while indexing performance is improved by the availability of more accurate density estimates, memory requirements should not continually increase. The WVQ clustering algorithm introduced in Chapter 6 was used to successfully demonstrate this exact behavior on a database of object models. Because there are essentially two degrees-of-freedom in the viewpoint variation of a feature grouping, the distribution of each one is approximately a 2D manifold embedded in a higher-dimensional index space (modulo noise and perspective effects). There is thus some reason to believe that the complexity of these distributions is limited, and that the number of prototypes required to represent them may be fairly small.

8.2 Limitations of the Approach

Specific object models: This approach is designed to recognize specific instances of objects rather than generic objects. This restriction allows accurate probabilities to be determined for direct image-to-model match hypotheses, but means that a much larger set of objects may have to be represented within the database in order that all members of a class may be identified. Some extra degrees-of-freedom (e.g., articulation) can be handled without changing the indexing framework, simply by increasing the number of training views to cover all aspects of the variation.

Incomplete search: Because the system is probabilistic, it is always possible that the indexing mechanism may fail. In contrast to, for example, a complete search of an interpretation tree, by limiting groupings to certain special types we have removed from consideration the vast majority of model-image feature set matches. In fact, ignoring these low-probability hypotheses is a critical aspect of our method which leads to the observed efficiency. Other probabilistic aspects of the search for matches are (i) the grouping algorithm itself, which can fail to detect groupings of the specified types, and (ii) the approximate nearest neighbour search algorithm.

The likelihood of overall recognition failure can be made extremely small by ensuring that each view of an object contains several feature groupings, so that any image of the object includes a few valid groupings useful for indexing.

8.3 Future Work

Probability estimates: There are several avenues to explore which could improve the accuracy of the probability estimates used for ranking. Modelling the background distribution of image shapes will be important if the model database does not provide a set of feature groupings representative of the global distribution. As suggested in Section 3.6.2, representing the background as a sparse set of prototypes with large interpolating kernels would provide a slowly-varying density estimate, and even modelling it as a constant level could improve performance. Having background distributions available would mean that information from different grouping types could be meaningfully combined. Without these, probability estimates for a grouping type are actually *relative* measures of saliency within the model database, which are useful for generating rankings within the same type. Given the background levels, estimates become *absolute* a posteriori estimates which can be compared between grouping types. The fact that background levels for more complex shapes would be smaller than those for simple shapes corresponds to, and could be used to implement, the intuitive notion that complex shapes are less ambiguous.

More work could be done to optimize various parameters in the estimation methods. For a given database, cross-validation could be used to determine the best value for k in the kNN and WkNN approaches. The VSM results indicated that global optimization of the distance metric and kernel size was not very effective. This may be because different values are appropriate in different regions of the index space. A method that is able to automatically partition the space and estimate parameters for each region would be a useful addition.

Feature groupings Extending the repertoire of grouping types will be important for increasing the robustness of the system. There are two ways to do this, both important: by increasing the number of different combinations of the current perceptual primitives (parallel pairs and coterminating pairs), and by adding other feature primitives from which groupings can be formed. Examples of additional primitives are those for curved image edges, color regions, and texture regions.

It will clearly be necessary to move beyond straight edges to include smooth image curves for indexing. There are two problems here: segmentation and representation. Curves can be broken at points of maximum and minimum curvature or at inflection points, and curve sections such as concavities can also be used as units of shape. Once segmented, there are many possible representations to consider, from simple circular or elliptical arcs to splines, snakes, etc. The solutions to both problems are much less stable to noise and scaling than with straight edges, and much work will be required to determine the most appropriate choices for integration with the proposed indexing method. The process of smoothing curves at multiple scales can be used to mitigate stability problems. Fortunately, our algorithm allows for some failure in feature detection by demanding redundancy of indexable groupings for each model view, and noise is well handled by the interpolating properties of the approach. In fact, it may be possible to capture the variation of curved feature values with noise, etc., with the same machinery used to model the variation of shape with viewpoint, and to generate probability distributions by sampling real image curves.

Some interesting special case invariants exist for color features, which have the potential to be combined with the current set of geometric features. For example, Slater & Healey (1995) use planar surface patches with matte reflectance characteristics to generate invariants of color histograms. They require the assumption that the reflectance function is a linear model in a small number of parameters. Problems with their method include that the size of the projected region changes with viewpoint, and that a variety of window sizes must be applied at all positions over the entire image. Combining their method with ours solves both issues. By associating

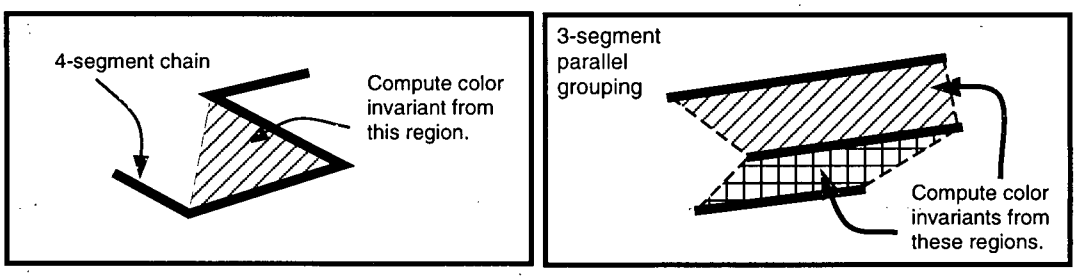


Figure 8.1: Regions to use when combining color invariants method of Slater and Healey with the current indexing function method, for two kinds of perceptual feature grouping.

color regions with perceptual feature sets (see Figure 8.1), the geometric information would allow normalization of the size of the image region, and the number of regions to test would be dramatically reduced.

A second example is Nayar & Bolle (1993), which calls for a slowly-varying surface and narrow-band sensors. For a given wavelength λ_0, the reflectance function is separated into a geometrical function and a reflectance coefficient:

$$r(\vec{\mathbf{s}}, \vec{\mathbf{v}}, \vec{\mathbf{n}}, \lambda_0) = \rho_0 R(\vec{\mathbf{s}}, \vec{\mathbf{v}}, \vec{\mathbf{n}}).$$

If the neighbouring points have similar scattering functions R but different reflectance coefficients ρ, then the ratio of measured brightnesses at neighbouring points should be invariant to illumination conditions and to imaging geometry. Therefore, this will be a useful invariant in cases where a surface consists of a single material, but the albedo, or color, changes from place to place.

Nayar and Bolle perform reflectance ratio indexing using groupings of three segmented color regions. To combine their method with our own, the average reflectance ratio across an edge could be used as an extra feature, added to current feature vectors (see Figure 8.2).

Memory usage: In order to minimize memory requirements, it may be useful to perform an analysis of the distributions of feature groupings. A recent paper does this for pairs of angles (Malik & Whangbo 1997), but knowledge of the form of the distributions of more complex groupings would provide information that could be used to limit the number of prototypes

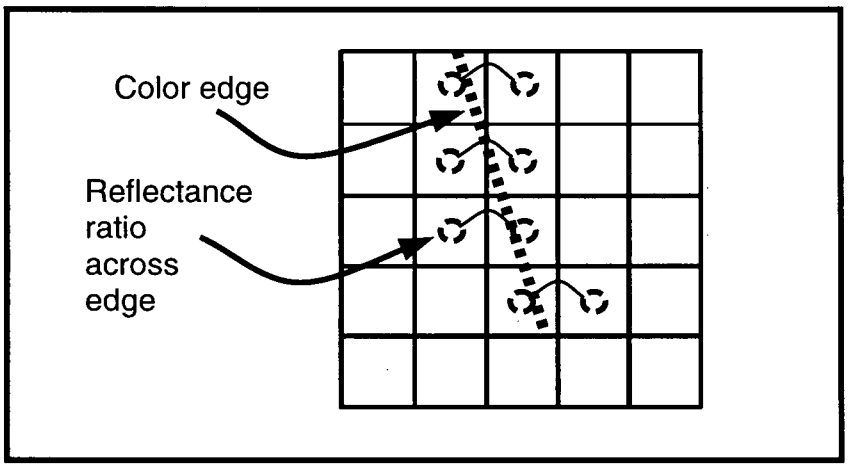


Figure 8.2: To integrate method of Nayar and Bolle with the current indexing function approach, reflectance ratio feature should be associated with single segments instead of whole regions. The single feature of average reflectance ratio over the length of the segment could be used. Only those segments involved in perceptually significant groupings, such as coterminating segment chains and parallel segment groupings, would need to have reflectance ratios computed for them.

allocated to represent them.

Modifying object models: There is currently no way to infer the presence of additional model features directly from image data. Rather, each feature must be specified by the user. A simple approach would be to save all unaccounted-for image features adjacent to a recognized object, and to correlate information over many recognitions. However, the amount of memory required for this procedure might be impractical.

Verification: Finally, the verification stage could easily be improved by the use of a probabilistic match extension process, to handle ambiguous feature matches. It would not be difficult to formulate a measure of ambiguity so that, early in the verification procedure only unambiguous matches would be added. Once the pose estimate stabilized, match tolerances could be tightened eliminating many of the ambiguities, and ensuring that any late inclusions would not degrade the pose.

Bibliography

Arya, S. (1995). Nearest neighbor searching and applications, *Technical Report CAR-TR-777*, Center for Automation Research, University of Maryland.

Ayache, N. & Faugeras, O. (1986). Hyper: a new approach for the recognition and positioning of two-dimensional objects, *IEEE Trans. PAMI* **PAMI-8**(1): 44–54.

Babich, G. & Camps, O. (1996). Weighted parzen windows for pattern classification, *IEEE Trans. PAMI* **18**(5): 567–570.

Barrett, E., Payton, P., Haag, N. & Brill, M. (1991). General methods for determining projective invariants in imagery, *CVGIP* **53**(1): 46–65.

Beis, J. (1996). Building models with planar faces using a structure-from-motion algorithm plus a small amount of post-processing, *Technical Report TR-96-14*, University of British Columbia.

Beis, J. & Lowe, D. (1994). Learning indexing functions for 3-d model-based object recognition, *Proceedings CVPR '94*, Seattle, Washington, pp. 275–280.

Beis, J. & Lowe, D. (1997). Shape indexing using approximate nearest-neighbour search in high-dimensional spaces, *Proceedings CVPR '97*, San Juan, Puerto Rico, pp. 1000–1006.

Ben-Arie, J. (1990). The probabilistic peaking effect of viewed angles and distances with application to 3-d object recognition, *IEEE Trans. PAMI* **12**(8): 760–774.

Bezdek, J. (1980). A convergence theorem for the fuzzy isodata clustering algorithms, *IEEE Trans. PAMI* **PAMI-2**(1): 1–8.

Biederman, I. (1985). Human image understanding: recent research and a theory, *CVGIP* **32**: 29–73.

Botros, S. & Atkeson, C. (1991). Generalization properties of radial basis functions, *Neural Computation* **3**: 579–588.

Broomhead, D. & Lowe, D. (1988). Multivariable functional interpolation and adaptive networks, *Complex Systems* **2**: 321–355.

Bruckstein, A., Holt, R., Netravali, A. & Richardson, T. (1993). Invariant signatures for planar shape recognition under partial occlusion, *CVGIP: Image Understanding* **58**(1): 49–65.

Burns, J., Weiss, R. & Riseman, E. (1993). View variation of point-set and line-segment features, *IEEE Trans. PAMI* **15**(1): 51–68.

Califano, A. & Mohan, R. (1994). Multidimensional indexing for recognizing visual shapes, *IEEE Trans. PAMI* **16**(4): 373–392.

Canny, J. (1986). A computational approach to edge detection, *IEEE Trans. PAMI* **PAMI-8**(6): 679–698.

Chen, C. & Kak, A. (1989). A robot vision system for recognizing 3-d objects in low-order polynomial time, *IEEE Trans. SMC* **19**(6): 1535–1563.

Clemens, D. & Jacobs, D. (1991). Space and time bounds on indexing 3-d models from 2-d images, *IEEE Trans. PAMI* **13**(10): 1007–1017.

Devijver, P. & Kittler, J. (1980). On the edited nearest neighbor rule, *5th Int. Conf. Patt. Rec.*, pp. 72–80.

Dickinson, S., Pentland, A. & Rosenfeld, A. (1992a). 3-d shape recovery using distributed aspect matching, *IEEE Trans. PAMI* **14**(2): 174–198.

Dickinson, S., Pentland, A. & Rosenfeld, A. (1992b). From volumes to views: an approach to object recognition, *CVGIP: Image Understanding* **55**(2): 130–154.

Duda, R. & Hart, P. (1973). *Pattern Classification and Scene Analysis*, Wiley, New York.

Fisher, R. (1936). The use of multiple measurements in taxonomic problems, *Annals of Eugenics* **7**: 179–188.

Flynn, P. & Jain, A. (1992). 3d object recognition using invariant feature indexing of interpretation tables, *CVGIP: Image Understanding* **55**(2): 119–129.

Forsyth, D., Mundy, J., Zisserman, A. & Brown, C. (1990). Invariance - a new framework for vision, *Proceedings ICCV '90*, pp. 598–605.

Franke, R. (1982). Scattered data interpolation: tests of some methods, *Mathematics of computation* **38**(157): 181–200.

Friedman, J., Bentley, J. & Finkel, R. (1977). An algorithm for finding best matches in logarithmic expected time, *ACM Trans. Math. Software* **3**: 209–226.

Fukunaga, K. (1990). *Introduction to Statistical Pattern Recognition*, second edn, Academic Press, Boston.

Fukunaga, K. & Hayes, R. (1989). The reduced parzen classifier, *IEEE Trans. PAMI* **PAMI-11**(4): 423–425.

Fukunaga, K. & Mantock, J. (1984). Nonparametric data reduction, *IEEE Trans. PAMI* **PAMI-6**(1): 115–118.

Gates, G. (1972). The reduced nearest neighbor rule, *IEEE Trans. Inf. Theory* pp. 431–433.

Gersho, A. (1979). Asymptotically optimal block quantization, *IEEE Trans. Inform. Theory* **IT-25**: 373–380.

Glassner, A. S. (ed.) (1990). *Graphics Gems*, Academic Press, San Diego, CA.

Gool, L. V., Kempenaers, P. & Oosterlinck, A. (1991). Recognition and semi-differential invariants, *Proceedings CVPR '91*, pp. 454–460.

Grimson, W. & Huttenlocher, D. (1990). On the sensitivity of geometric hashing, *Proceedings 3rd ICCV*, pp. 334–338.

Grimson, W. & Lozano-Perez, T. (1987). Localizing overlapping parts by searching the interpretation tree, *IEEE Trans. PAMI* **PAMI-9**(4): 469–482.

Hart, P. (1967). The condensed nearest neighbor rule, *IEEE Trans. Inf. Theory* pp. 515–516.

Hartman, E., Keeler, J. & Kowalski, J. (1990). Layered neural networks with gaussian hidden units as universal approximators, *Neural Computation* **2**: 210–215.

Huttenlocher, D. & Ullman, S. (1987). Object recognition using alignment, *Proceedings ICCV '87*, pp. 102–111.

Jacobs, D. (1991). Optimal matching of planar models in 3d scenes, *Proceedings CVPR '91*, pp. 269–274.

Jaynes, E. (1957). Information theory and statistical mechanics, *Physical Review* **106**(4): 620–630.

Kim, B. & Park, S. (1986). A fast k nearest neighbor finding algorithm based on the ordered partition, *IEEE Trans. PAMI* **PAMI-8**(6): 761–766.

Lamdan, Y., Schwartz, J. & Wolfson, H. (1990). Affine invariant model-based object recognition, *IEEE Trans. Rob. Aut.* **6**(5): 578–589.

Lamdan, Y. & Wolfson, H. (1988). Geometric hashing: a general and efficient model-based recognition scheme, *Proceedings ICCV '88*, pp. 238–249.

Lloyd, J., Beis, J., Pai, D. & Lowe, D. (1997). Model-based telerobotics with vision, *Proceedings ICRA '97*, Albuquerque, New Mexico, pp. 1297–1304.

Lloyd, J. & Hayward, V. (1992). Multi-rccl user's guide, *Technical Report Technical Report*, Center for Intelligent Machines, McGill University.

Lowe, D. (1985). *Perceptual organization and visual recognition*, Kluwer Academic, Hingham, MA.

Lowe, D. (1987a). Three-dimensional object recognition from single two-dimensional views, *Artificial Intelligence* pp. 355–395.

Lowe, D. (1987b). The viewpoint consistency constraint, *Int. J. Computer Vision* **1**(1): 57–72.

Lowe, D. (1991). Fitting parametrized three-dimensional models to images, *IEEE Trans. PAMI* **13**(5): 441–450.

Lowe, D. (1995). Similarity metric learning for a variable-kernel classifier, *Neural Computation* **7**(1): 72–85.

Malik, R. & Whangbo, T. (1997). Angle densities and recognition of 3d objects, *IEEE Trans. PAMI* **19**(1): 52–57.

Miclet, L. & Dabouz, M. (1983). Approximative fast nearest neighbor recognition, *Pattern Recognition Letters* **1**: 277–285.

Mohan, R., Weinshall, D. & Sarukkai, R. (1993). 3d object recognition by indexing structural invariants from multiple views, *Proceedings ICCV '93*, pp. 264–268.

Nayar, S. & Bolle, R. (1993). Reflectance ratio: a photometric invariant for object recognition, *ICCV '93*, pp. 280–285.

Neimann, H. & Goppert, G. (1988). An efficient branch-and-bound nearest neighbour classifier, *Pattern Recognition Letters* **7**: 67–72.

Nene, S. & Nayar, S. (1996). Closest point search in high dimensions, *Proceedings CVPR '96*, pp. 859–865.

Omohundro, S. (1989). Geometric learning algorithms, *Technical Report TR-89-041*, ICSI.

Pathak, A. & Camps, O. (1993). Bayesian view class determination, *IEEE CVPR '93*, pp. 407–412.

Poggio, T. & Girosi, F. (1990). Regularization algorithms for learning that are equivalent to multilayer networks, *Science* **247**: 978–982.

Rigoutsos, I. (1992). *Massively parallel Bayesian object recognition*, PhD thesis, Courant Institute of Mathematical Sciences, NYU.

Rose, K., Gurewitz, E. & Fox, G. (1990). Statistical mechanics and phase transitions in clustering, *Phys. Rev. Lett.* **65**(8): 945–948.

Rose, K., Gurewitz, E. & Fox, G. (1993). Constrained clustering as an optimization method, *IEEE Trans. PAMI* **15**(8): 785–794.

Rothwell, C., Zisserman, A., Mundy, J. & Forsyth, D. (1992). Efficient model library access by projectively invariant indexing functions, *Proceedings CVPR '92*, pp. 109–114.

Ruck, D., Rogers, S., Kabrisky, M., Oxley, M. & Suter, B. (1990). The multilayer perceptron as an approximation to a bayes optimal discriminant function, *IEEE Trans. Neur. Net.* **1**(4): 296–298.

Saund, E. (1993). Identifying salient circular arcs on curves, *CVGIP: Image Understanding* **58**(3): 327–337.

Shashua, A. (1993). On geometric and algebraic aspects of 3-d affine and projective structures from perspective 2-d views, *Technical Report 1405*, MIT AI Lab.

Shimoji, S. & Lee, S. (1994). Data clustering with entropical scheduling, *Int. Conf. Neural Networks*, Vol. 4, pp. 2423–2428.

Shimshoni, I. & Ponce, J. (1995). Probabilistic 3d object recognition, *Proceedings ICCV '95*, pp. 488–493.

Slater, D. & Healey, G. (1995). Combining color and geometric information for the illumination invariant recognition of 3-d objects, *ICCV '95*, pp. 563–568.

Smith, S. (1993). *Geometric optimization methods for adaptive filtering*, PhD thesis, Harvard Univ.

Sproull, R. (1991). Refinements to nearest-neighbor searching in k-dimensional trees, *Algorithmica* **6**: 579–589.

Stein, F. & Medioni, G. (1992a). Structural indexing: efficient 2-d object recognition, *IEEE Trans. PAMI* **14**(12): 1198–1204.

Stein, F. & Medioni, G. (1992b). Structural indexing: efficient 3-d object recognition, *IEEE Trans. PAMI* **14**(2): 125–145.

Taylor, C. & Kriegman, D. (1995). Structure and motion from line segments in multiple images, *IEEE Trans. PAMI* **17**(11): 1021–1032.

Thompson, D. & Mundy, J. (1987). Three-dimensional model matching from an unconstrained viewpoint, *Proceedings Int. Conf. Rob. Aut.*, pp. 208–220.

Tsai, R. (1987). A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses, *IEEE J. Robotics and Automation* pp. 323–344.

Turney, J., Mudge, T. & Volz, R. (1985). Recognizing partially occluded parts, *IEEE Trans. PAMI* **PAMI-7**(4): 410–421.

Wallace, A. (1987). Matching segmented scenes to models using pairwise relationships between features, *Image Vis. Comput.* **5**(2): 114–120.

Wan, E. (1990). Neural network classification: a bayesian interpretation, *IEEE Trans. Neur. Net.* **1**(4): 303–305.

Wayner, P. (1991). Efficiently using invariant theory for model-based matching, *Proceedings CVPR '91*, pp. 473–478.

Weinshall, D. (1993). Model-based invariants for 3-d vision, *Int. J. Comp. Vis.* **10**(1): 27–42.

Weinshall, D. & Tomasi, C. (1993). Linear and incremental acquisition of invariant shape models from image sequences, *Proceedings ICCV '93*, pp. 675–682.

Weiss, I. (1988). Projective invariants of shapes, *Proceedings DARPA Image Understanding Workshop*, pp. 1125–1134.

Weiss, I. (1992). Noise resistant projective and affine invariants, *Proceedings CVPR '92*, pp. 115–121.

Wheeler, M. & Ikeuchi, K. (1995). Sensor modelling, probabilistic hypothesis generation, and robust localization for object recognition, *IEEE Trans. PAMI* **17**(3): 252–265.

Witkin, A. & Tenenbaum, J. (1983). On the role of structure in vision, *in* Beck, Hope & Rosenfeld (eds), *Human and Machine Vision*, Academic Press, New York, pp. 481–543.

Xie, Q., Laszlo, C. & Ward, R. (1993). Vector quantization technique for nonparametric classifier design, *IEEE Trans. PAMI* **15**(12): 1326–1330.

Appendix A

A.1 Hidden-Line Removal Complexity

Let F_i be the number of faces for model i, and S_i the number of segments. In the training stage, the HLR algorithm tests all visible segments against all visible faces to determine the hidden lines. (About half of the faces and segments face away from the camera and are not visible. These are quickly determined from the normal vectors of the faces, and are not considered here.) The complexity of HLR is simply

$$(\#segments) \times (\#faces) \tag{A.1}$$

$$\approx (\frac{1}{2}S_i) \cdot (\frac{1}{2}F_i). \tag{A.2}$$

The training stage can be used to determine which segments are occluded by which faces, from which viewpoints. At runtime, this information is recovered for the closest pose, and used to speed up the HLR process. Although the value may be different for different models, there will be an average reduction in the number of faces that will have to be checked for each visible segment, and complexity will be reduced to

$$\approx (\frac{1}{2}S_i) \cdot (\frac{1}{2}fF_i). \tag{A.3}$$

This can be tested empirically to see how well this "effective" fraction of occluding faces characterizes the process. Figure A.1 shows that the relations above are followed very closely in practice. Each point represents one of the database models, having a unique combination

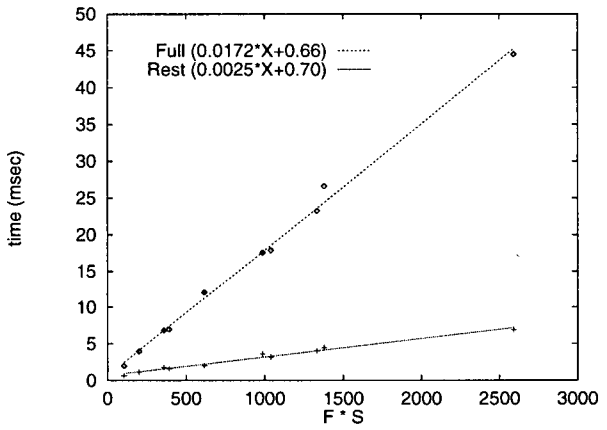


Figure A.1: Hidden-line removal timing for database of 10 models.

of faces F and segments S. The times shown are times for HLR averaged over at least 100 projections from different camera viewpoints. The ratio of the slopes gives the fraction f:

$$\frac{0.0025}{0.0172} = \frac{1}{6.88}. \tag{A.4}$$

A.2 Corner Detection Complexity

For a square image, of length L in each dimension and containing S segments, the density of segment endpoints is

$$D_e = \frac{2 \times \#segments}{area\ of\ image} = \frac{2S}{L^2}. \tag{A.5}$$

The number of coterminations for a given endpoint will be the number of endpoints from other segments found in a circle of radius R about the endpoint in question. This is simply the area of the circle times the density of endpoints in the image:

$$(\pi R^2)(\frac{2S}{L^2}) \tag{A.6}$$

To achieve scale independence with these groupings, the search radius R should be some fraction of the segment length. If the average segment length is l, then the average search radius

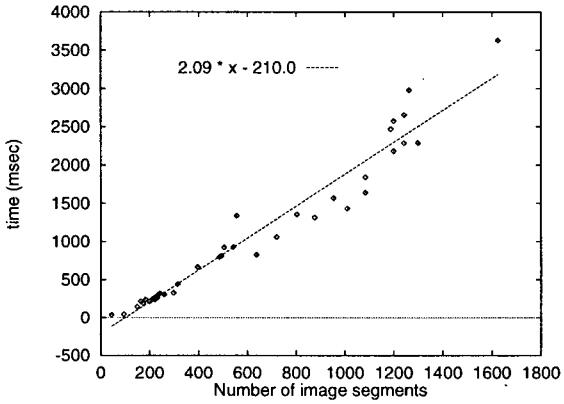


Figure A.2: Corner detection timing for several images (linear fit to points).

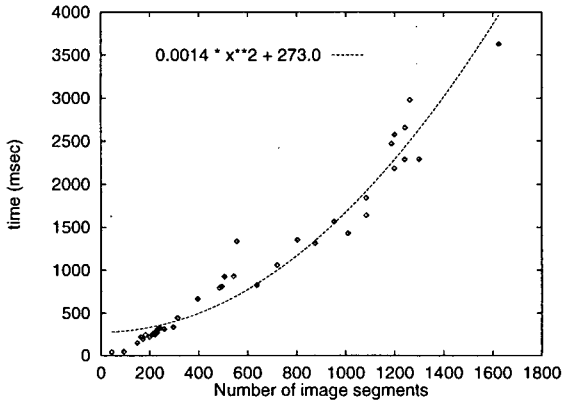


Figure A.3: Corner detection timing for several images (quadratic fit to points).

will be $R = k_1 \cdot l$. The average number of coterminations to a given endpoint is then

$$\langle C_1 \rangle = (\pi R^2)(\frac{2S}{L^2}) = 2\pi S(\frac{k_1 l}{L})^2 \tag{A.7}$$

The expected complexity for forming cotermination groupings is the total number of image endpoints times the average number of coterminations per endpoint:

$$\begin{aligned} \langle C \rangle &= 2S \times \langle C_1 \rangle && \text{(A.8)} \\ &= (2S)(2\pi S)(\frac{k_1 l}{L})^2 && \text{(A.9)} \\ &= 4\pi S^2(\frac{k_1 l}{L})^2 && \text{(A.10)} \\ &\propto S^2 && \text{(A.11)} \\ &= \mathcal{O}(S^2) && \text{(A.12)} \end{aligned}$$

This is perhaps intuitively obvious, and basically says that each segment must be compared against some fraction of the remaining segments for this task. Timing experiments for images of various sizes, i.e., with different numbers of segments, are presented in Figure A.2 and Figure A.3. The best quadratic fit is given, and a linear fit for comparison. A similar analysis holds for parallel groupings.

A.3 Time Complexity of Indexing

Define:

N_m: Number of models.

N_p: Average number of stored points per model.

N_t: Total number of stored points ($= N_m N_p$).

N_i: Number of image features.

N_g: Number of image groupings ($= \mathcal{O}(N_i)$, from Chapter 3).

k: Number of nearest neighbours to recover for each image grouping.

S: "Search-multiple", i.e. examine at most $S \cdot k$ leaf nodes when searching the kd-tree for nearest neighbours.

Standard Search

In our method (see Chapter 5), an approximate search is used to recover nearest neighbours from the kd-tree. In this method, there is a maximum number of leaf nodes examined for each query, which is taken to be some small multiple S of the number k of neighbours to recover. Using the standard kd-tree search (as opposed to BBF search), the maximum lookup time per query is therefore

$$(\#NN)(\frac{\#leaves}{NN})(\frac{access\ time}{leaf}) = (k)(S)(\log N_t) \tag{A.13}$$

where the access time is logarithmic in the number of stored points. The total lookup time T_l for all queries is thus

$$T_l = (\#queries)(\frac{time}{query}) = (N_g)(kS \log N_t). \tag{A.14}$$

What is hidden in this expression is that k is not fixed, but must vary with N_m and N_p as the size of the index increases. In fact, we will assume that N_p has some practical maximum value that can be fixed, so that $N_m \propto N_t$. (N_p essentially corresponds to the number of views required to represent the object. This number may be quite small, as suggested in Figures 4.13

and 4.14. How to choose an absolute value for k is problematic, as noted in Section 4.4.4, but we can explore how it must change with N_m. To see that $k = \mathcal{O}(N_m)$, note that k essentially fixes a small volume within which the kNN or WkNN probability estimate is computed. Now suppose that N_m is increased while N_p remains fixed. Since increasing the number of models stored should not change which of the previously stored models are recovered as hypotheses (i.e. assuming that k has been set properly to begin with), we need the kNN volume to (on average) remain the same. This implies $k \propto N_m$, or $k \propto N_t$.

Using this proportionality for k:

$$T_l = \mathcal{O}(N_g \cdot kS \cdot \log(N_t)) \tag{A.15}$$

$$T_l = \mathcal{O}(N_i N_t \log(N_t)). \tag{A.16}$$

Index lookup time with standard kd-tree search is therefore *linear* in the number of image features and *log-linear* in the number of object models, in the worst case.

BBF Search

In BBF search, in addition to the work done by the standard search, there is a small amount of overhead required to maintain a priority queue (See Chapter 5). The access time for a leaf node includes two components of work, (a) the time to traverse internal nodes of the tree all the way to the leaf node, at each step inserting into the priority queue, and (b) the time to extract the next search path from the queue.

If the priority queue is implemented as a heap, both insertion and extraction require $\mathcal{O}(L_q)$ time, where L_q is the current length of the queue. Because the entire search for any query looks at no more than $k \cdot S$ leaves, the length of the queue can be no greater than

$$L_q \leq (\#\textit{leaves visited})(\textit{max tree height}) \tag{A.17}$$

$$L_q = (kS)(\log N_t). \tag{A.18}$$

In the worst case, after a leaf is examined the next path in the search is back at the top of the tree, and $\mathcal{O}(\log N_t)$ internal nodes must be traversed each time. The total traversal time

T_t is thus

$$T_t = (a) + (b) \leq (\log N_t)(\log L_q) + \log L_q \quad (A.19)$$

$$= (\log N_t + 1)(\log L_q). \quad (A.20)$$

Since the maximum number of traversals for one query is kS, the total work for a single query is

$$(\frac{time}{query}) = (\frac{\#\ traversals}{query})(\frac{time}{traversal}) = (kS)(T_t) \quad (A.21)$$

$$\leq (kS)(\log N_t + 1)(\log L_q) \quad (A.22)$$

and the total indexing, or lookup, time is

$$T_l = (\#\ queries)(\frac{time}{query}) \quad (A.23)$$

$$\leq (N_g)(kS)(\log N_t + 1)(\log L_q) \quad (A.24)$$

$$\leq (N_g)(kS)(\log N_t + \log N_t)(\log L_q) \quad (A.25)$$

$$= (N_g)(kS)(2 \log N_t)(\log L_q) \quad (A.26)$$

$$\leq (N_g)(kS)(2 \log N_t)(\log[kS \log N_t]) \quad (A.27)$$

$$= \mathcal{O}(N_i \cdot N_t \cdot \log N_t \cdot \log[N_t \log N_t]) \quad (A.28)$$

$$\leq \mathcal{O}(N_i N_t \log N_t \log[N_t^2]) \quad (A.29)$$

$$= \mathcal{O}(N_i N_t \log N_t \log[N_t]) \quad (A.30)$$

$$= \mathcal{O}(N_i N_t \log^2 N_t). \quad (A.31)$$

As with the expression for standard search, if we assume that N_p has some practical maximum value, N_t can effectively be thought of as the number of object models. The index lookup time with BBF search is then *linear* in the number of image features, and scales as $N_m \log^2 N_m$ with the number of object models (since $N_m \propto N_p$).