Indexing Without Invariants in 3D Object Recognition

Jeffrey S. Beis and David G. Lowe
Computer Science Department
University of British Columbia

This paper has been published in *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **21,** 10 (1999), pp. 1000–1015.

Abstract

We present a method of indexing 3-dimensional objects from single 2-dimensional images. Unlike most other methods to solve this problem, ours does not rely on invariant features. This allows a richer set of shape information to be used in the recognition process. We also suggest the kd-tree as an alternative indexing data structure to the standard hash table. This makes hypothesis recovery more efficient in high-dimensional spaces, which are necessary to achieve specificity in large model databases. Search efficiency is maintained in these regimes by the use of Best-Bin First search, a modified kd-tree search algorithm which locates approximate nearest-neighbours. Neighbours recovered from the index are used to generate probability estimates, local within the feature space, which are then used to rank hypotheses for verification. On average, the ranking process greatly reduces the number of verifications required. Our approach is general in that it can be applied to any real-valued feature vector. In addition, it is straightforward to add to our index information from real images regarding the true probability distributions of the feature groupings used for indexing. In this paper, we provide experiments with both synthetic and real images, as well as details of the practical implementation of our system, which has been applied in the domain of telerobotics.

1 Introduction

In general terms, an index is a reorganization of a set of data, from its natural ordering to one in which certain search tasks become easier. In object recognition the data are shapes, or appearances, and the rearrangement is from a database of labelled object models to an ordering by numerical representation of shape. In almost all approaches to date, the data structure used to store the reordered data has been the hash table, and most of the interest has focused on which type of special-case invariant shape descriptor to store there. In this paper, we apply indexing to *non-invariant* features, and argue that there must be a reconsideration of the best way to reorder the shape data, regardless of the exact shape representation chosen. To this end we suggest the

kd-tree, with a modified kd-tree search algorithm, as a more efficient alternative to the ubiquitous hash-table data structure.

At first glance, invariant features and hash tables appear to be the optimal choices for indexing efficiency. Since invariant features by definition do not change with viewpoint, a single index entry could be used to represent a feature, or set of features, from all viewing directions, and memory requirements would be minimized. As well, hash tables promise constant time lookup of entries, which would mean the time complexity could not be further reduced.

It has been shown [1, 2, 3], however, that no invariants exist for general 3D point sets. Because of this, many methods have developed special-case invariants by placing constraints on feature sets, for example, that all features lie in a plane [4, 5, 6, 7, 8]. We believe this approach is too restrictive for the 3D-from-2D recognition problem, and has meant that a wealth of shape information from non-invariant groupings has been unavailable to the indexing mechanism.

Hash tables also turn out to be problematic when one considers indexing in high-dimensional spaces. This regime is important as it corresponds to the use of more complex shapes, which are necessary to provide the high degree of specificity required to discriminate between a large number of object models. The main 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 means finding a set of *nearest-neighbours* (NN) to a query point. It turns out that nearest-neighbour search in a hash table requires a time exponential in the dimension of the space. The problem lies in the fact that the closest neighbour may not lie in the same hash bin as the query point, but rather in one of the many adjacent bins.

Our two-stage indexing process consists of an off-line training stage and a run-time recognition stage. In the training stage, synthetic images of each object are acquired by sampling from the view sphere. The images are processed to extract feature groupings. In this paper, two types of feature grouping, called Local Feature Sets, are used for indexing: chains of coterminating segments, and parallel segment groupings. Angle and edge-length relationships between the segments of a grouping are used to generate a multi-dimensional feature vector, which can be stored in the index structure. Since feature appearance changes with viewpoint, and our features are non-invariant, a given underlying 3D grouping will in general be represented by a small cluster of points within the index space. These are samples from the probability distribution of the underlying grouping, and given enough samples, what will effectively be stored in the index are accurate estimates of these distributions.

At runtime, feature vectors are extracted from the test image and used to access the kd-tree, recovering a small set of nearest-neighbour model vectors. An approximate NN search method, called *Best Bin First* search [9], is used to extend the efficiency of this search to higher-dimensional spaces, at the expense of occasionally missing one of the actual closest neighbours. We then employ a kernel-based probability estimator, which relies on the number and distance of neighbours indicating a particular underlying grouping. While interpolating between the pre-stored training views recovered in the NN search, the estimator provides the likelihood that a given hypothesis is correct. Ranking hypotheses according to their probabilities provides the best ordering for a subsequent verification stage, increasing the accuracy of indexing (also noted by [10, 11]). Note that in cases where invariant features are available, they can be easily included within our framework, with the attendant reduction in storage.

2 Previous Work

We first discuss several prominent indexing methods which have used invariants and hash tables for indexing. Forsythe *et al.*[4], outlined several types of projective invariant feature for indexing planar objects viewed in arbitrary 3D orientations. Their experiments were carried out using a 2D index space generated by pairs of coplanar conic sections. Rothwell *et al.*[7], used 4D indices defined by area moments of planar curves, that were first transformed into canonical reference frames. In each of these methods, the dimensionality of the spaces and the number of models were too small for the inefficiency of hashing to be critical.

The *geometric hashing* technique ([6, 5]) has also generally been applied in low- (2- or 3-) dimensional index spaces generated from planar point sets. In this method, triples of points are used to establish bases in which the coordinates of other, coplanar points remain invariant. Models are stored redundantly in the index, using all combinations of coplanar points as bases and hashing the remaining points into the table. This means that at run-time, if a correct basis triple is chosen from among the image points, the corresponding model pattern is available and readily detected.

While this technique differs substantially from other indexing methods in that voting overcomes some of the difficulty with bin boundaries, Grimson [12] 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. Memory usage from the redundant storage of models is also excessive. In [5], geometric hashing was applied with larger (8D) indices generated from planar curves ("footprints"). However, the experiments did not truly test indexing performance because only a few models were used, with 3 or 4 features each.

Stein and Medioni presented a method suitable for 2D-from-2D [13] and 3D-from-3D [14] indexing. The novel aspects of this work included the use of multiple levels of smoothing to improve segmentation [13], and the introduction of a new feature type, the "splash", for non-polyhedral 3D data [14]. However, since the regimes they worked with give rise to invariant features, they were able to avoid the difficult issue of non-invariance associated with the 3D-from-2D problem. Hash tables were again used as the index structure. While the former paper 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 hash bins (e.g., 60° for angle features), and looking only in the single bin containing the query point. However, 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. Neither does it preclude missing a significant percentage of good hypotheses which lie just across one of the many bin boundaries. Beis and Lowe [15] provide evidence that, no matter how coarse the binning, hash table performance is poor in high-dimensional spaces.

In [16], Califano and Mohan argue for the use of higher-dimensional spaces in indexing. Their analysis indicates a dramatic reduction in recognition time by increasing the size of the feature vectors. However, they again use hash tables for the lookup and do not search across bin boundaries. 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. Because a number of separate groupings is required to initiate a hypothe-

sis, and each grouping is already non-robust due to being high-dimensional, their method will lead to a large number of false negatives when real images are used. Our experiments show that adequate discrimination can be achieved using significantly less information, namely a single high-dimensional feature grouping, but weighted according to its uniqueness or saliency.

The method of Clemens and Jacobs [3], while using hash tables, did not assume invariant feature sets. They derived 4D and 6D index spaces from general (i.e., non-planar) 3D point sets. However, the difficult task of feature grouping was done manually, and the model database and index spaces were again too small for hashing inefficiency to be noticeable.

The most similar work to our own is that of Dickinson *et al.* [17, 18]. Like us, they compute probabilities of association between non-invariant features observed in images and object models, but 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. Combinations of the small number of "top-level" primitives are used to generate a wide variety of possible object models. By using a fixed set of primitives, they are able to pre-compute the probabilities of association between features in any two layers of the hierarchy, and to store these in index tables. However, because the primitives are simple, with variable dimensions, the inferences from bottom- to top-level are very weak. In contrast, the use of more complex quantitative information as we suggest below leads to strong probabilistic inferences directly from the image data to the model database.

There have been a number of recent appearance-based approaches to object recognition, in which many sampled 2D views are used as the representation for 3D objects. Because these methods do not use 3D models, they do not solve for accurate object pose as in the approach given in this paper. However, these approaches do simplify the model acquisition process and they explore a number of new feature types that could prove useful for increasing the range and robustness of recognition. Nelson and Selinger [19] use a novel form of grouping called an image patch. A patch is formed from a curve segment bounded by points of high curvature, and all other curves within a square region around the curve are included in the grouping. This has the advantage that it can make use of curved and disconnected edge segments. Schmid and Mohr [20] have used a set of differential invariants computed at image corner points as key features for indexing. These have the advantage of being ubiquitous in natural images, but are more sensitive to illumination change than edge-based features. Other more global approaches to appearance-based recognition include eigenspace matching [21] and receptive field histograms [22], but their more global features make them more sensitive to clutter and partial occlusion.

3 The Problems of Shape-based Indexing

We next delineate several issues involved in the application of indexing to the problem of object recognition. Because the use of geometric features is much more developed in the recognition community than, say, color or texture, the association of "features" with "shape" will be more natural for most readers, so the term "shape" will be used rather than "appearance".

Problem 1: *Shape is ambiguous.* A given set of image features may suggest a match with more

Figure 1: A single image grouping (white lines) indicates several possible model matches (black lines) via indexing. At most one of these may be valid.

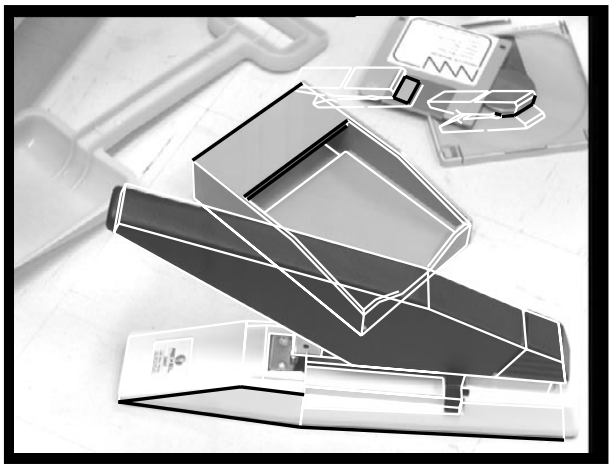

Figure 2: Several feature groupings detected in the image (black lines) indicate possible model matches (white lines) via indexing. In general, only a small percentage of these will be valid.

than one object (Figure 1). This is clear, as sub-parts of different objects may even be identical. The problem is especially relevant for the 3D-from-2D recognition problem, where a wide variety of 3D shapes can result in the same 2D shape after projection, depending on the viewpoint of the camera. Similarly, the image will contain many feature groupings, only a small number of which are likely to be valid (Figure 2). In both situations it is important for the indexing process to indicate which of the recovered hypotheses are more likely to be correct.

Problem 2: *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, the features 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 arise.

Problem 3: *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 can cause small variations in image feature values. Ideally, the shape descriptor would be invariant to the change in viewpoint, and only a single index entry would be needed for each 3D model shape, but as previously mentioned, invariant features are too restrictive for this problem. However, it will be seen that partial invariance is important for the same reason that complete invariance would be ideal.

Problem 4: *Data is imperfect.* For a variety of reasons, image data used as input to the indexing algorithm will not be perfect.

1. *Occlusion.* An object may be partially hidden behind another object.

2. *Noise.* Feature values may be slightly distorted from one image to another depending on lighting conditions, digitization effects, and segmentation algorithms.

3. *Mis-detection of features.* In the case of edges, a segmentation algorithm may break a single edge into several smaller edges, or confound edges from two (or more) objects, generating a single edge. 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.

Any indexing solution to the 3D-from-2D recognition problem must address each of these problems, while at the same time attending to the fundamental issues of time and space complexity. In the next section, we outline just such an approach.

4 Indexing Without Invariants

4.1 Introduction

As noted by Clemens and Jacobs [3] and others, the first stage of any indexing method must be feature grouping. This is because single features will generally not have sufficient specificity to differentiate between many different objects. We use local groupings of primitive features called Local Feature Sets, grouped according to certain perceptual properties (see below). These are represented as feature vectors in a multi-dimensional feature space. We choose groupings that are partially invariant to the full set of imaging transformations, keeping space requirements reasonable. Various combinations of the primitive features are used to develop a rich set of redundant groupings, as a counter to occlusion and missing features.

Our indexing stage is broken into two parts: search and probability estimation. Because it is impractical to store analytical forms for the probability distributions of the Local Feature Sets, we store samples of the underlying distributions. These are collected in a training stage, during which all objects in the database are imaged from many different viewpoints, and a feature extraction process is applied. The samples are stored in a kd-tree, which is the natural data structure for NN search in continuous, real-valued multi-dimensional feature spaces.

At runtime, the same feature extraction process is applied to the test image, generating image vectors used to query the index. The use of the kd-tree, in conjunction with BBF search (see below), provides rapid access to the closest stored neighbours, i.e., the shapes most similar to the detected image shapes. A kernel-based probability estimator uses the set of recovered neighbours, interpolating between nearby stored views of an object and handling the problem of viewpoint variation of shape. Smooth estimates are also useful to provide graceful degradation of performance in the presence of noise. The probabilities are used to form a ranked list of hypotheses. While there is no perfect way to differentiate all objects based on local descriptors of shape, the use of probability estimates deals effectively with the inherent ambiguity of indexing. Thus we can think of our index as containing probability distributions for certain types of feature set, and our two-stage indexing method as a way to access these distributions. A summary of the proposed indexing method is provided in Figure 3. The choices detailed above provide a comprehensive solution to all of the problems outlined in Section 3, and scale well with the size of the model database. This approach has been explored in detail by Beis [9].

Training stage:

1: Generate a set of training examples for each object by observing it from many points of view.

2: Store the associated feature vectors in the kd-tree data structure. There will be one kd-tree for each type of feature grouping.

Recognition stage:

1: Derive N feature vectors $\vec{\mathrm{x}}_\mathrm{i}$ from image.

2: Use BBF search to recover, for each $\vec{\mathrm{x}}_\mathrm{i}$, a set of k nearest-neighbour model vectors, indicating potential matches. The total number of unique hypotheses is thus $\leq k \cdot N$.

3: Use a local, kernel-based estimator to interpolate between nearby recovered training views, and compute probabilities for each unique match hypothesis as in Equation 2 or 3. (E.g., for kNN, $p(m|\vec{\mathrm{x}}) = k_m/k$.) Rank hypotheses for verification according to probability estimates.

4: Verify hypotheses, to eliminate any false positives generated by indexing.

Figure 3: Indexing methodology. The four stages of the approach (grouping, search, probability estimation, and verification) are described in more detail below.

4.2 Shape representation

The constraints on shape representation imposed by our indexing framework are minimal: any relatively stable, real-valued feature is appropriate. In this paper we have chosen to use perceptually significant groupings of straight edge segments as the fundamental set of shapes with which feature vectors are generated. Segment primitives are grouped together if they exhibit certain non-accidental properties such as parallelism, cotermination, or symmetry (only the former two have been implemented to date). It is important to note that many other types of primitive/grouping are possible, which could be used to extend the scope of the indexing approach.

Feature values are derived from relationships between the segments, and consist of angles and edge length ratios. These features are *partially* invariant to the full set of model-to-image

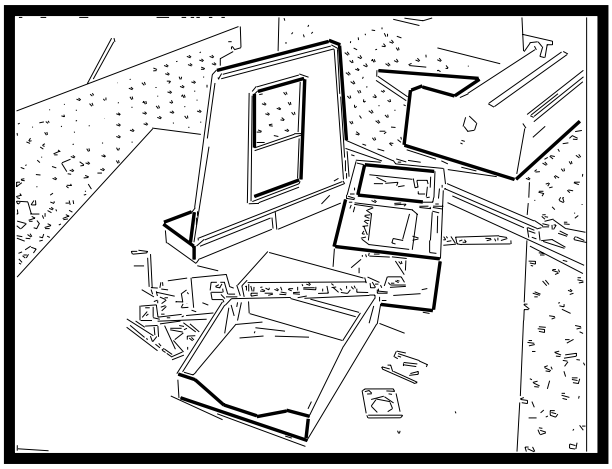

Figure 4: Examples of segment chain feature groupings (bold lines) of various cardinalities.

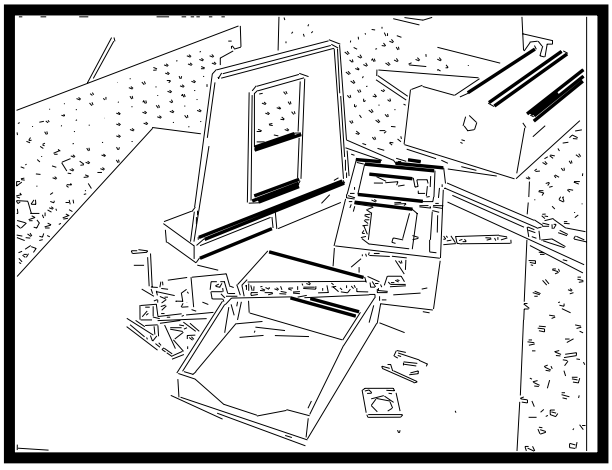

Figure 5: Examples of parallel feature groupings (bold lines) of various cardinalities.

transformations, i.e., to translation and image plane rotation[1], and only change significantly with out-of-plane rotations. This is important, as each invariant degree of freedom drastically reduces the number of training examples required to represent the probability distribution for a given grouping (see Section 4.4). So, while we reject the concept of totally invariant features as too restrictive, we believe that the proposed level of invariance is both very useful and straightforward to achieve with many different types of feature.

Examples of segment chain and parallel groupings are given in Figures 4 and 5. An example of a feature vector would be $(\theta_1, \theta_2, \theta_3, l_3/l_2)$ for a 4-segment chain, where θ_i are the angles between consecutive segments i and $i+1$, and l_3/l_2 is the length ratio of the two interior chain

[1]In fact, this is only true when using the weak perspective projection approximation. Indexing objects under large perspective distortions would require the use of extra training examples, to cover the additional range of shape variation.

segments. There is a practical trade-off between the *specificity* and the *robustness* of groupings. The larger a grouping, the less ambiguous the shape, but the more likely that the grouping will fail to be properly detected, due to occlusion, noise, etc.

Therefore, in order that the indexing process be accurate and at the same time not overly brittle, a large degree of redundancy is required. Any single view of an object should contain several heterogeneous groupings capable of generating vectors useful for indexing. While the larger groupings are more fragile, when they are detected they should lead to rapid recognition. In cases where they fail to be detected, the system can fall back on smaller, more ambiguous groupings, which will generally have lower probability estimates, hence lower rankings, and therefore require somewhat longer verification times.

An important issue for consideration with such groupings is the relative scaling of different dimensions in the feature space. Each dimension refers to a separate quantity, and each may be measured in different units. Because the notion of distance within the feature space is so important for indexing, the distance must be made meaningful with respect to shape similarity.

For the purely geometric features used in this paper, we have set the scaling by hand, taking into account expected noise levels and intuitive notions of similarity (see Section 4.4 for further details). For more complex combinations of features, an optimization method such as Variable-kernel Similarity Metric learning [23] may be important to determine the best values for the dimensionality weighting parameters.

4.3 Index Structure

The primary goal of indexing must be rapid runtime recovery of good hypotheses. We formulate this as a k nearest-neighbour (kNN) search in which, for a given query point (i.e., image feature vector), a small number of the closest points in the index space are recovered. This is in contrast to hash table methods, which effectively do a range search, recovering all points within a given distance from the query point.

By using kNN search, as opposed to range search, we sacrifice a small amount of thoroughness for a large gain in efficiency. This is because, while it is true that most of the neighbours recovered in a range search are possible matches, the vast majority will have a very small probability of being valid. In general, the highest probability hypotheses can be discovered by observing just a few of the closest neighbours.

It is well known that tree structures are more efficient than arrays for retrieving NN in higher-dimensional spaces. One of the most widely used of these is the kd-tree [24] [25]. (Note that the Rtree, a structure similar to the kd-tree but useful for range searches, requires extra storage that becomes impractical as the dimensionality increases.) The kd-tree is built as follows. 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 until a bin contains fewer than a small, fixed number of points. This creates a balanced binary tree with depth $d \leq \lceil \log_2 N \rceil$, with the equality holding when the number of points per bin is 1.

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

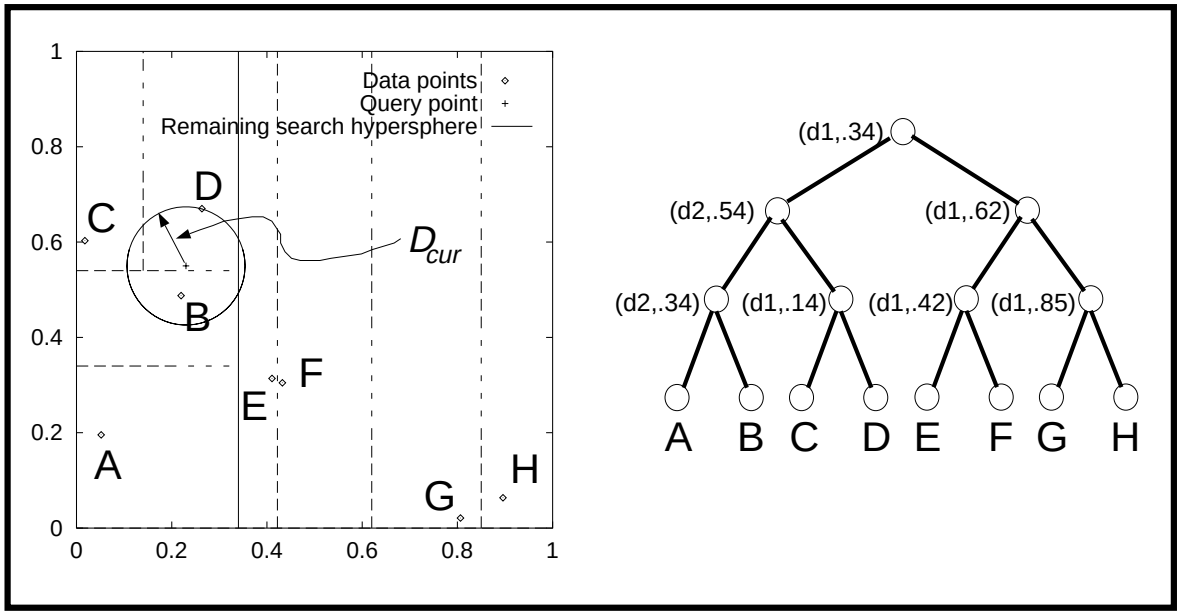

Figure 6: kd-tree with 8 data points labelled A-H, dimension of space k=2. On the right is the full tree, the leaf nodes containing the data points. Internal node information consists of the dimension of the cut plane and the value of the cut in that dimension. On the left is the 2D feature space carved into various sizes and shapes of bin, according to the distribution of the data points. The two representations are isomorphic. The situation shown on the left 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.

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 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, hence the claim in the original paper of logarithmic time lookup of NN. Unfortunately, as dimensionality increases, the same problem that occurs with hash tables happens here (albeit to a much lesser extent), and the search algorithm ends up examining too many adjacent bins [26]. Much of this time, however, is spent testing bins unlikely to contain the best neighbour. An *approximate* search algorithm which ignores most of these low-probability bins, *Best Bin First* (BBF) search [15], can be used to extend the practical range of the method to much higher dimensionalities.

The basic idea behind BBF search is to look in the bins closest in Euclidean distance to the query point first, rather than those closest in the kd-tree branch structure (the latter being the

tactic used in the standard kd-tree search — see Figure 6). BBF search expands outward from q in a more symmetric fashion, and makes it more likely that the true NN will be encountered early on in the search.

This property leads to the other improvement involved in BBF, which is simply to cut off the search after examining a fixed number of leaf nodes. It turns out that a high percentage of the time the algorithm has located the best neighbour, but doesn't know it yet. In these cases, terminating the search early has no negative effect.

BBF requires no change in the construction stage of the kd-tree, and the runtime search can be easily implemented with a small amount of overhead using a priority queue. 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 to hold 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.

While the kd-tree data structure adapts itself during off-line construction to the locations of the stored points, BBF adapts the runtime search to the position of the query point. As shown in [15], for moderate dimensionalities (e.g., 8-15), and for large numbers of stored points, the BBF algorithm uncovers the exact NN a high percentage of the time, and a very close neighbour in the remaining cases. This is true even when only a small number of leaf nodes are examined. Making NN search efficient in higher dimensions allows for the use of more information in the feature vectors, and thus for more accurate indexing results.

In contrast, hash tables are in the uncomfortable position of trying to determine a single, optimal bin size, when data density may vary dramatically from place to place within the space. If the hash table search is limited to a single bin, as has often been the case [13, 16], then no matter how coarsely a high-dimensional space is divided, the vast majority of NN will be missed. For example, suppose we have the following scenario (see [15]): an 8-dimensional space with a uniform, random distribution of stored points (best case for hashing) in a unit hypercube. Suppose further that each dimension is coarsely divided into 4 equal bins, and the number of points is chosen to give 1 $point/bin$, i.e. $4^8 = 62536$. Then a single-bin hash table search will return only about 18% of the nearest neighbours. On the other hand, expanding the hash table search to include neighbouring bins makes it exponential in the dimension of the index space.

4.4 Estimating Probabilities

The fact that we allow non-invariant features, together with the continuous or real-valued nature of most features, means it is necessary to represent the continuous set of 2D appearances of 3D feature groupings. Various analyses [27, 2] have shown that the probability that a 3D feature will take on one of its possible 2D appearances is non-uniform, and in many cases will be sharply peaked about a particular value. Ideally, we would determine concise analytical forms for these distributions, and hence have accurate probability information stored in a compact form. However, such expressions are extremely complex even for pairs of features [28], while we would like to use high-dimensional feature vectors. Computing the analytical forms also requires knowledge of the *a priori* likelihood of viewing each object from a particular viewpoint.

The alternative, which we have chosen, is to sample object appearance from several viewpoints about each object. Initially, this can be done either from random viewpoints, or using some evenly-spaced tessellation of the view sphere. A local, kernel-based method is then used to interpolate between samples, and provide the probability estimates necessary for accurate indexing. While this approach initially requires the same assumption as the method above (equal *a priori* probability for all viewpoints), it is much less complex, especially as grouping size increases. Furthermore, the assumptions can be relaxed once the system is in operation, by adding information from real images back into the index, to update the probabilities according to the true rate of appearance of model feature vectors.

We have investigated two of the standard non-parametric density estimators, the k nearest-neighbours (kNN) and Weighted kNN (WkNN) methods. Using the notation of [29], the density for class m at point $\vec{\mathbf{x}}$ can be estimated using kNN as

$$p(\vec{\mathbf{x}}, m) = \frac{k_m/n}{V} \tag{1}$$

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:

$$p(m|\vec{\mathbf{x}}) = \frac{p(\vec{\mathbf{x}}, m)}{\sum_{c=1}^{C} p(\vec{\mathbf{x}}, c)} = \frac{k_m}{k} \tag{2}$$

Similarly, if the neighbours are weighted by a window, or kernel, function, we have the WkNN estimate

$$p(m|\vec{\mathbf{x}}) = \frac{p(\vec{\mathbf{x}}, m)}{\sum_{c=1}^{C} p(\vec{\mathbf{x}}, c)} = \frac{\sum_{i=1}^{k_m} \phi(\|\vec{\mathbf{x}} - \vec{\mathbf{x_i}}\|/\sigma)}{\sum_{i=1}^{k} \phi(\|\vec{\mathbf{x}} - \vec{\mathbf{x_i}}\|/\sigma)}. \tag{3}$$

In this paper we have used the Gaussian kernel

$$\phi(\frac{\|\vec{\mathbf{x}}\|}{\sigma}) \equiv \frac{1}{\sqrt{2\pi}\sigma} e^{-(\vec{\mathbf{x}}\cdot\vec{\mathbf{x}}/2\sigma^2)} \tag{4}$$

but any smooth window function should give similar results. WkNN weights each recovered point according to its proximity to the test vector, which intuitively suggests a more accurate estimate than kNN, with only a slight increase in computation.

One problem for both methods is the choice of k. A larger value for k means that noise is reduced and estimates are smoother, but that higher frequency variations in the distributions may be lost. In our experiments, we have used a small value ($k = 10$) to keep the estimates local, and this provided satisfactory performance.

In the WkNN approach there is a further parameter, the window width σ which, like k, affects the smoothness of the interpolation. The window width is closely coupled with the requirement to set the relative weighting of each dimension. The weightings determine distances within the multi-dimensional feature space where different dimensions refer to qualitatively different properties. As it is not independent of the weights, it is acceptable to fix σ at a value of 1.0, and set

only the weights relative to σ. These have been determined according to what we believe the indexing algorithm should consider "similar" shapes. For example, for dimensions representing angular features, a difference of (σ =) 0.3 radians, or 17 degrees, was weighted to give a distance of 1.0 in distance calculations. This means that a neighbour at distance 0 is weighted by a value 1.0, while a neighbour at distance 17° is given a weighting of $e^{-(0.3 \cdot 0.3/2\sigma^2)} = e^{1/2} = 0.61$. The other feature type used in this paper, edge-length ratio, was given the (unitless) weighting $\sigma = 0.5$.

Our use of inexact matching means that our method can handle small amounts of noise as well as slight deformations in the dimensions of the actual object. In order to deal with large variations in object shape, such as with a parametrized object (e.g. hinged or telescoping), it would not be appropritate to simply relax the criteria of similarity by changing the dimensionality weightings. Rather, the number of training views would need to be increased to cover the increased variation in shape evident in such an object.

These methods are simple yet effective, as our experiments below demonstrate. Note that a few of our parameters have been set manually, including k (number of NN to recover), the BBF cutoff (maximum number of leaf nodes to visit during the search), and the dimensionality weightings. We have found results to be relatively insensitive to the exact operating point of these parameters, and that for a given database, a reasonable operating point may be determined with a small amount of experimentation (e.g. see Section 5.1.2 for number of training views required). The exact parameter settings are much less important for ensuring good indexing performance than is the specificity of the feature groupings, i.e. the dimensionality of the index vectors. Fortunately, small increases in grouping size lead to large increases in specificity.

Finally, we note that because probabilities are computed at runtime, based on a set of nearest-neighbours recovered from the index, updating probabilities based on new information simply requires adding new points into the index structure. Further details regarding index update procedures from real image data, and the associated clustering methods used to keep space requirements low, are available in [9].

4.5 Verification

Given the estimated probabilities, hypotheses are sorted into a list ranked according to likelihood of being correct. Beginning with the most likely hypothesis, we perform a verification procedure which consists of two alternating steps. The first is an iterative pose determination based on the current match set. The second step attempts to extend the match set to include additional object-image feature pairs.

Pose is computed using an iterative, least-squares algorithm due to Lowe [30]. At each stage, the algorithm minimizes the perpendicular distances of projected model segment endpoints from image segments. The range of convergence for this method is quite large, and we have not observed any problem with the pose estimates available from the recovered nearest-neighbour samples (i.e., for true positive hypotheses). In general, few iterations are required before the algorithm converges.

Given the computed pose estimate, the model can be back-projected into the image to search for further matches. In early iterations while the pose is still approximate, one procedure that has proven to be very important in the match extension process is that the algorithm only matches

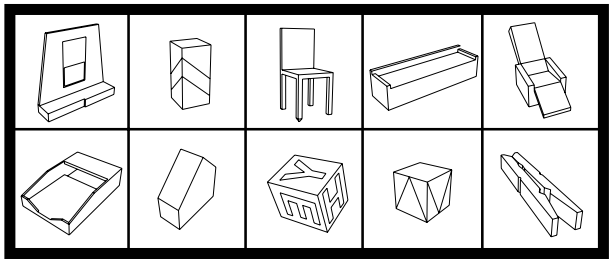

Figure 7: Model database.

model segments which are adjacent on the model to segments in the current match set. Because the initial pose may be inaccurate, model segments far from the current match set may project to image positions quite distant from their correct image match, and by coincidence get matched with a "clutter" segment. This often causes verification to fail altogether, but if not, it leads to inaccuracy in the final pose estimate.

Matches were considered valid if greater than 50% of the lengths of visible model segments were matched. More sophisticated procedures are required to make recognition robust, using information such as the number and density of image features, number of matches, etc. Because of this, in order to better separate performance of the indexing mechanism from that of the entire recognition algorithm, we have used both manual and automatic verification in many experiments, and provide comparisons of the results of both.

5 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, we investigate the degree to which each component can be said to work, and provide estimates of the time required for each one.

5.1 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 fails. The database of 10 object models is shown in Figure 7. The variables we investigate are (i) 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, depending on the experiment. Grouping methods were

applied to each image and the resulting training vectors stored in the (kd-tree) index. An independent set of test images was then generated, where each was of a single database object chosen at random and viewed from a random viewpoint. Because some objects had two-fold symmetry, test views were restricted to a hemisphere; otherwise, a valid match to a symmetric counterpart grouping could 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. The number of nearest-neighbours to retrieve, k, was set to 10, and the maximum number of stored points examined per query was 100. For a single run, each data point was generated by averaging over 100 test images, consisting of 10 random images of each of the 10 database objects. Error bars (standard deviations) were produced by repeating the whole process 10 times with different random seeds.

Two performance measures were used:

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

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

5.1.1 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 two modes of probability estimation were compared.

The two methods are virtually indistinguishable with respect to the percent found (Figure 8). 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 are virtually identical in ranking the hypotheses (Figure 9), and both curves show 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.

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

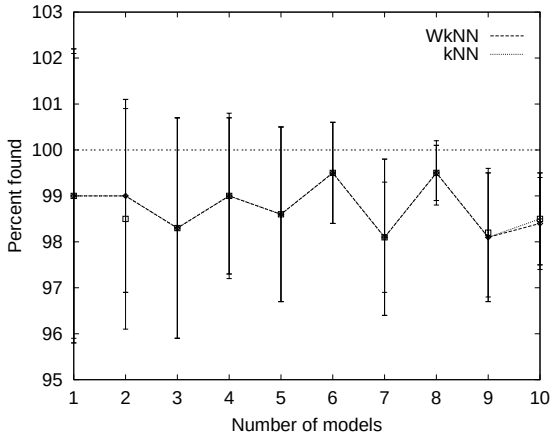

Figure 8: Performance vs. Number of models: Percent found.

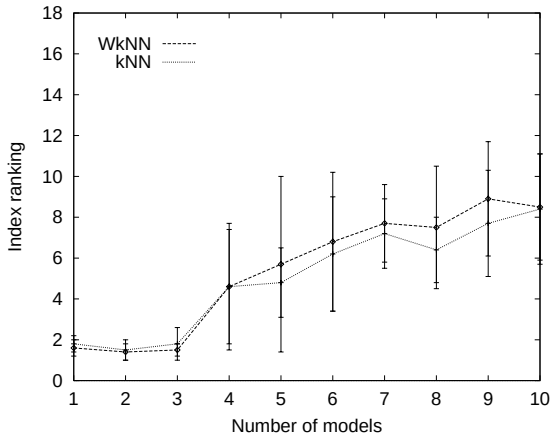

Figure 9: Performance vs. Number of models: Index ranking of the first correct hypothesis. Lower numbers are better.

5.1.2 Number of Views

In this experiment: training views were randomly sampled over a hemisphere; 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 (Figure 10) 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 prototypes are necessary to accurately represent the probability distributions of most groupings.

The improvement in rankings with number of training views (Figure 11) means that the probability estimates are working as expected. For most other methods, storing a larger number of distractors in the index would mean *worse* performance, because more hypotheses would be generated.

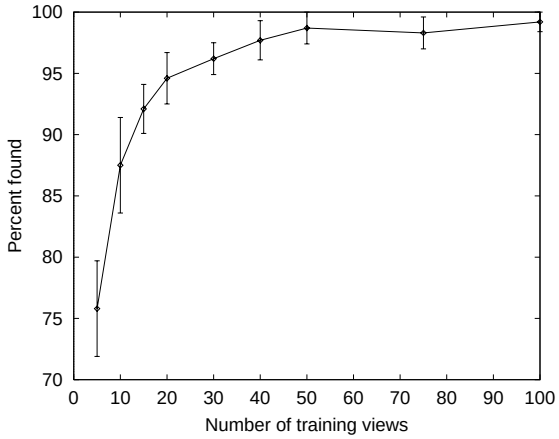

Figure 10: Performance vs. Number of views: Percent found.

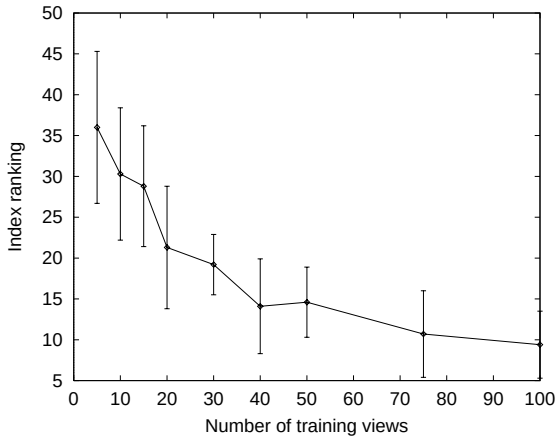

Figure 11: Performance vs. Number of views: Index ranking of the first correct hypothesis. Lower numbers are better.

5.1.3 Dimension of Index Space

In this experiment: tessellated training was used (240 views over sphere); the WkNN mode of probability estimation was applied; 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 (Figure 12) 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 may therefore have failed for some of the more widely spaced (in index space) views. The solution to this is to index using several types of grouping in parallel. In case recognition fails with the more specific larger groupings, the system can fall back on lower-dimensional indexes.

The decrease in ranking with index space dimensionality (Figure 13) 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.

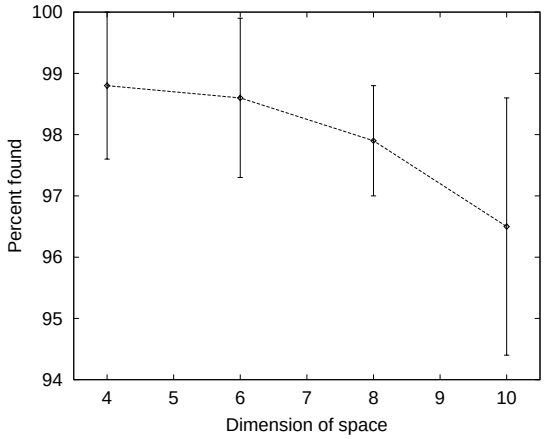

Figure 12: Performance vs. Dimension of index space: Percent found.

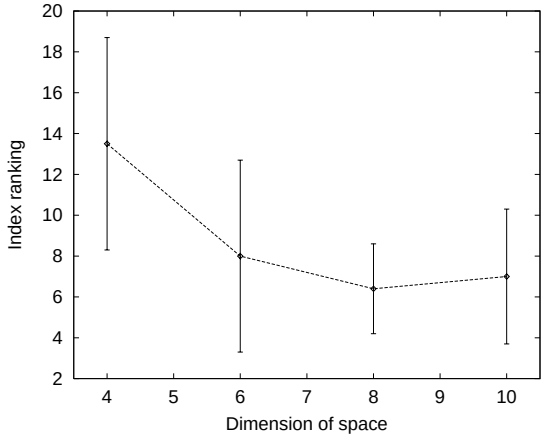

Figure 13: Performance vs. Dimension of index space: Index ranking of the first correct hypothesis. Lower numbers are better.

5.1.4 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 4 pixels. 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 (Figure 14) and the slow increase in ranking (Figure 15) both indicate a graceful degradation with increasing noise, due to the availability of "smooth" probability estimates for the indices. Noise moves a point about in feature space by a small amount, distorting 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 worse for smaller objects.

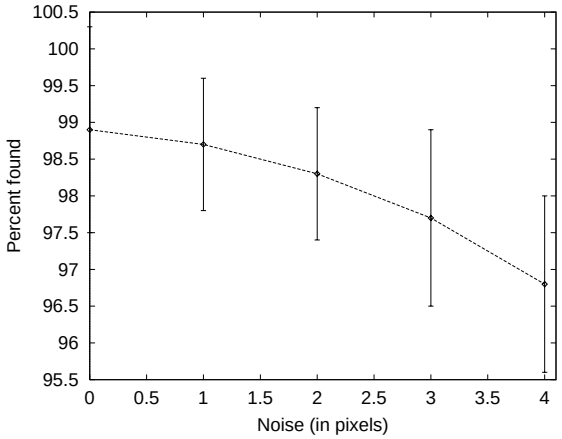

Figure 14: Performance vs. Image noise level: Percent found.

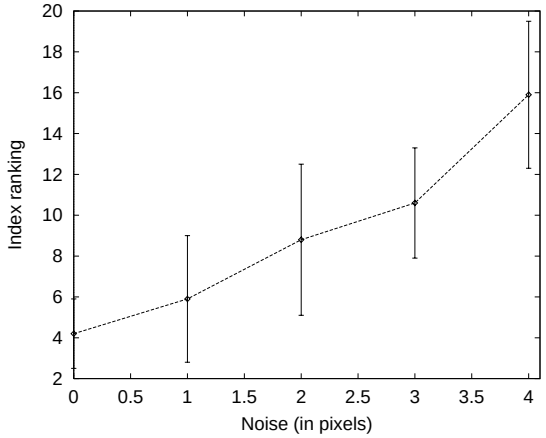

Figure 15: Performance vs. Image noise level: Index ranking of the first correct hypothesis. Lower numbers are better.

5.2 Real Images

It is very difficult to perform quantitative analyses of recognition performance with real images, as there is no precise, agreed-upon measure of image difficulty. In fact, the difficulty of a given image for recognition will vary according to the algorithm used. This makes it problematic to compare even the most similar current indexing methods, with researchers using different types of image data and different model representations.

For these reasons, we have organized our experiments with real images in a partly qualitative, partly quantitative manner. We loosely define three qualitative regimes, for which example images are presented, and within which quantitative experiments were performed. 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 16). The "moderate" regime has images with significant clutter, moderate occlusion, and a variety of object sizes (Figure 17). Finally, "difficult" images are those containing significant clutter, moderate

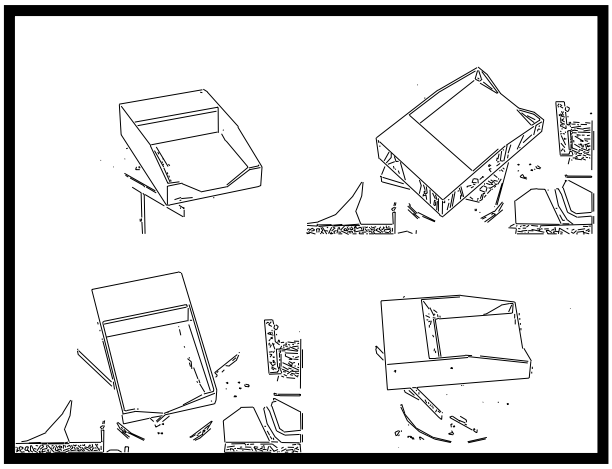

Figure 16: Four examples of "easy" images.

to significant occlusion, smaller objects, and a very large number of image features (Figure 18).

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 were used in the experiments below. 3-segment parallel groupings, while not in themselves highly effective for indexing, were found to be quite useful for augmenting the chain groupings. The 240-view tessellated training method was used for each of the 10 models. This provided a database of $211,000$ vectors from $13,600$ groupings for the 4-segment chains; $133,000$ vectors from 5500 groupings for the 3-segment chains; and $20,000$ vectors from 2000 groupings for the 3-segment parallel groupings. The vectors from each flavour of grouping were stored in separate kd-trees.

In the recognition stage we checked performance using the "optimal" grouping type (4-chains), and using all three types in concert, which is slower by approximately a factor of three. The same search parameters were used for BBF as in the synthetic image experiments, i.e., $k = 10$ nearest-neighbours, and no more than 100 points were examined per query. To isolate the indexing component from verification, we first checked each hypothesis manually for validity, before applying a simple automatic verification procedure for comparison (Section 4.5). While the verification issue certainly cannot be ignored, it is much more critical that an indexing method avoid false negatives (from which recovery is impossible) than it is to avoid false positives (which are expected and which a sophisticated verification module should detect), and manual verification allows us to separate the two for reporting.

5.2.1 Recognition with "Easy" Images

The difficulties introduced by moving from synthetic images even to very simple real images are non-trivial, so we consider this regime an important test for any indexing method. Complications introduced by lighting conditions and feature detection include: extra edges (from reflections on some object surfaces), missing edges (due to lack of contrast over the extent of an edge), broken edges (from lack of contrast in a local area, or due to a segmentation error), and combined

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

Table 1: 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. Further explanation in text.

edges (two distinct edges erroneously linked by the segmentation algorithm). In this section, 208 images were acquired of one of the models in isolation. Examples are shown in Figure 16.

Table 5.2.1 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 those 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 $200,000$ stored 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 (6) was there a need to explore more than 10 hypotheses before the object was found.

One advantage of our manual verification procedure was that it allowed for a detailed analysis of the failure modes of the automatic procedure. 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 (which exists in the weak perspective regime of imaging). This is easily correctable by trying both possible orientations (tilted either 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 match between two closely-spaced parallel segments. This problem is also easily corrected, by delaying ambiguous match decisions until after the pose has been well constrained using other, more certain matches. The remaining 2 verifications which failed were due to a combination of reasons.

5.2.2 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

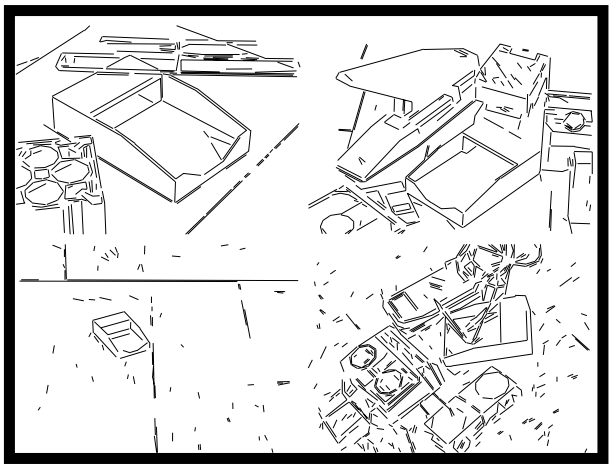

Figure 17: Four examples of images with "moderate" difficulty.

various sizes. Examples are shown in Figure 17. 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 before the object was identified, 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. The two successful indices that were not properly verified corresponded to the failure mode of correctly signalling object presence, but providing an inaccurate final pose.

5.2.3 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 more likely to disrupt the indexing mechanism. Occlusion is also a significant factor in many of these images, which are more complex than those attempted in 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

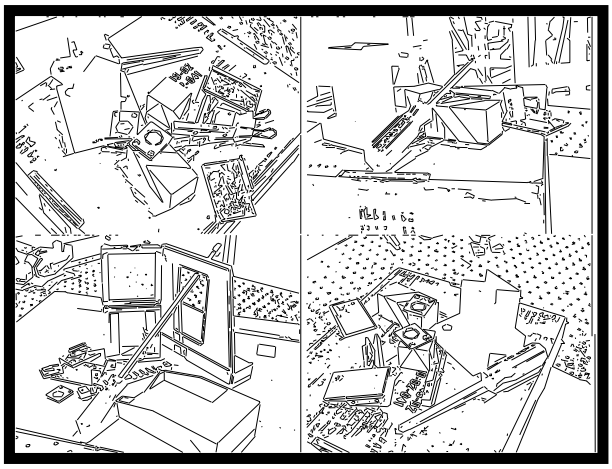

Figure 18: Four examples of "difficult" images.

median rank demonstrates that the probability estimates are indeed worthwhile, as in most cases no more than 50 of the 1000 hypotheses had to be tested before recognition succeeded.

5.2.4 Summary

In this section we have demonstrated strong indexing performance on the two sets of images that we qualitatively describe as "easy" and "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 the "indexing without invariants" methodology is 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 two straightforward additions to the verification procedure, i.e., dealing with the 2-fold degeneracy of planar groupings, and handling ambiguous feature 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 require larger groupings for discrimination, although the specificity of 4-segment chains with respect to the current database, by the use of probability estimates, provides evidence that extremely large groupings may not be required. The excellent scaling behavior of the BBF search algorithm to higher dimensionalities suggests that our approach is very promising for use with a significantly larger number of models.

5.3 Timing

We now look at the second important aspect of indexing, the time required to recover hypotheses from the index. For the purposes of this analysis, the recognition process can be subdivided into the three stages of grouping, indexing, and verification. Some definitions are required:

1. Let G be the average time to perform grouping, given a set of image line segments.
2. 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.
3. Let V be the average time to verify a single match hypothesis against the image.
4. Let N_i be the number of image groupings.
5. Let N_m be the number of underlying, 3D model groupings that can be used to form hypotheses.
6. Let k be the number of neighbours to recover from the index during BBF search.

We can look at recognition time in both worst-case and average-case estimates. The total time required will be the sum of the times for the three major components, grouping, indexing, and verification. Grouping time is the same in both cases, simply G. Indexing time for both will also be identical: 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 $N_i \times I$.

For the third component, verification, the worst case occurs if each recovered neighbour leads to a different hypothesis, and all hypotheses must be investigated. This leads to $N_i \times k$ 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 (call it A) will be much smaller than the worst case. Examples of typical values for A under various imaging conditions are given in the previous sections (the "index ranking" values), for synthetic and real images.

The expressions for total time are therefore:

$$\textbf{Average case:} \quad (G + N_i \times I + A \times V) \tag{5}$$

$$\textbf{Worst case:} \quad (G + N_i \times I + N_i \times k \times V) \tag{6}$$

The average observed time for grouping 4-segment chains in the "difficult" images (Section 5.2.3), which contained 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 were: $N_i = 1000$, $N_m = 10,000$ (for a 10-model database), $k = 10$, and $A = 10$ (average value for images of "moderate" difficulty).

These values lead to the following absolute time estimates:

$$\textbf{Average case:} \quad 2.9sec \tag{7}$$

$$\textbf{Worst case:} \quad 250sec \tag{8}$$

Stated times are for ANSI C code running on a SUN UltraSPARC-2 workstation. The above timing estimates were typical of those observed during system operation.

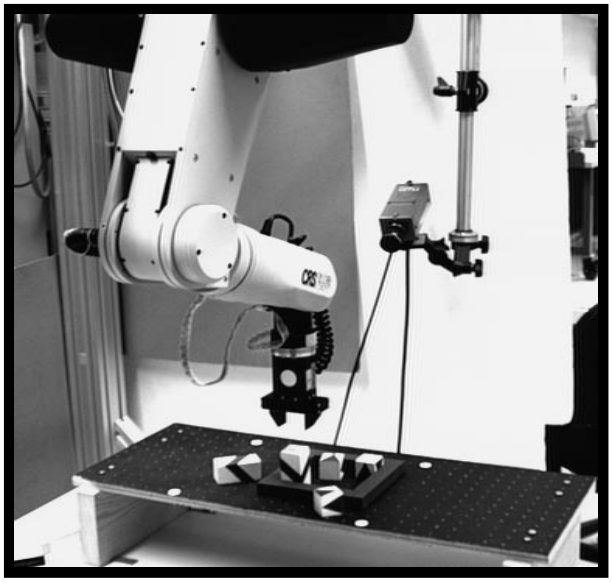

Figure 19: Remote site, showing the robot, camera, and work area.

5.4 Application to Telerobotic System

In this section, we describe the application to a real working system, summarizing the results of Lloyd *et al.*[31]. A 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 both to determine object presence, and also to provide accurate positional information, so that the robotic arm was able to "pick-and-place" the recognized objects.

Model-based Telerobotics is robotics that allows a user to manipulate objects at a distance, by interacting with a local graphical model of the remote site (Figure 19) rather than with the remote site itself. In our implementation, the local model (see Figure 20) is initialized and updated by the indexing and recognition system at the remote site. The model consists mainly of a 3D graphical representation of certain simple objects (wooden blocks) that are to be grasped by the robot arm. The positions and orientations of the objects relative to the camera and robot arm are accurately depicted. Note that, having the recognition system send only the coordinates of pertinent objects rather than entire video frames from the remote site has the potential to dramatically reduce the bandwidth required for the remote communication.

The local operator performs operations such as clicking on a particular object, moving the graphical object around the model workspace with a simulated robot arm and, once satisfied with the final positioning, sending a verification signal to the remote site causing the robot to physically perform the move. (Of course, the automatic handling of these high-level commands by the remote site requires some involved calculations, made possible by the use of the Robotic Control C Library (RCCL) [32].)

The implementation of the vision system required about 5 seconds per frame for recognition (SGI Indy workstation). This included a fast software version of the Canny edge detector

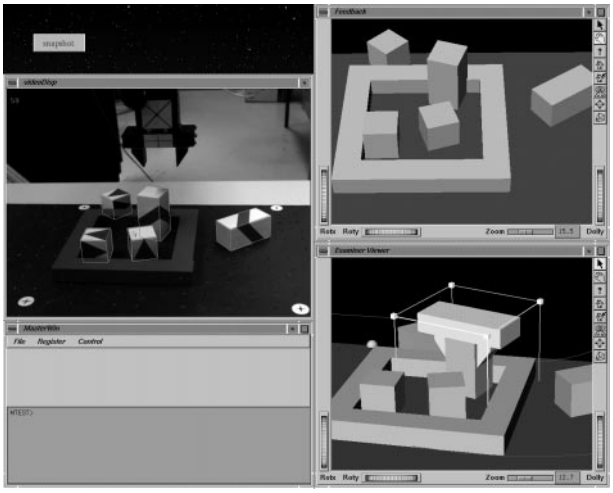

Figure 20: Screen image at operator site. Left window: Camera image window. Upper right window: Feedback model window, showing the latest model of the remote site, as given by the vision module. Lower right window: Object manipulation window. This is periodically initialized to the state of the feedback model window, but then is changed according to operator interactions.

[33], 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. Because the pose is far more precise in the directions parallel to the image plane than towards and away from the camera, accuracy in the latter 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. As noted previously, a more sophisticated verification procedure would be an important addition to the system, but for this demonstration extra constraints were used instead: verified object instances outside the physical range of the robotic arm were removed from consideration. Full details of results are available in Lloyd *et al.*[31].

6 Conclusion

We have presented a method of indexing 3D objects from single 2D images. It is novel in that it does not rely on invariant features, but instead uses stored samples of the distributions in feature space to form smooth probability estimates that a given shape corresponds to a particular database object. We presented experiments which showed that ranking hypotheses based on our estimates drastically reduces the number of verifications required. The efficient approximate nearest-neighbour search algorithm we apply to the kd-tree index structure keeps indexing efficient to higher dimensionalities than other approaches. This allows for the use of more complex groupings, which means that the system scales well with the number of stored models. Important

future work includes extending the repertoire of feature groupings, and incorporating an effective way to collect probability information from large sets of real images for inclusion into the index.

References

[1] E.B. Barrett, P.M. Payton, N.N. Haag, and M.H. Brill, "General methods for determining projective invariants in imagery," *CVGIP*, vol. 53, no. 1, pp. 46–65, 1991.

[2] J.B. Burns, R.S. Weiss, and E.M. Riseman, "View variation of point-set and line-segment features," *IEEE Trans. PAMI*, vol. 15, no. 1, pp. 51–68, January 1993.

[3] D.T. Clemens and D.W. Jacobs, "Space and time bounds on indexing 3d models from 2d images," *IEEE Trans. PAMI*, vol. 13, no. 10, pp. 1007–1017, October 1991.

[4] D. Forsyth, J.L. Mundy, A. Zisserman, and C.M. Brown, "Invariance - a new framework for vision," in *Proceedings ICCV '90*, 1990, pp. 598–605.

[5] Y. Lamdan, J.T. Schwartz, and H.J. Wolfson, "Affine invariant model-based object recognition," *IEEE Trans. Rob. Aut.*, vol. 6, no. 5, pp. 578–589, 1990.

[6] Y. Lamdan and H.J. Wolfson, "Geometric hashing: a general and efficient model-based recognition scheme," in *Proceedings ICCV '88*, 1988, pp. 238–249.

[7] C.A. Rothwell, A. Zisserman, J.L. Mundy, and D.A. Forsyth, "Efficient model library access by projectively invariant indexing functions," in *Proceedings CVPR '92*, 1992, pp. 109–114.

[8] P.C. Wayner, "Efficiently using invariant theory for model-based matching," in *Proceedings CVPR '91*, 1991, pp. 473–478.

[9] J.S. Beis, *Indexing without invariants in model-based object recognition*, Ph.D. thesis, University of British Columbia, June 1997.

[10] M.D. Wheeler and K. Ikeuchi, "Sensor modelling, probabilistic hypothesis generation, and robust localization for object recognition," *IEEE Trans. PAMI*, vol. 17, no. 3, pp. 252–265, March 1995.

[11] I. Shimshoni and J. Ponce, "Probabilistic 3d object recognition," in *Proceedings ICCV '95*, 1995, pp. 488–493.

[12] W.E.L. Grimson and D.P. Huttenlocher, "On the sensitivity of geometric hashing," in *Proceedings 3rd ICCV*, 1990, pp. 334–338.

[13] F. Stein and G. Medioni, "Structural indexing: efficient 2d object recognition," *IEEE Trans. PAMI*, vol. 14, no. 12, pp. 1198–1204, December 1992.

[14] F. Stein and G. Medioni, "Structural indexing: efficient 3d object recognition," *IEEE Trans. PAMI*, vol. 14, no. 2, pp. 125–145, February 1992.

[15] J.S. Beis and D.G. Lowe, "Shape indexing using approximate nearest-neighbour search in high-dimensional spaces," in *Proceedings CVPR '97*, San Juan, Puerto Rico, June 1997, pp. 1000–1006.

[16] A. Califano and R. Mohan, "Multidimensional indexing for recognizing visual shapes," *IEEE Trans. PAMI*, vol. 16, no. 4, pp. 373–392, April 1994.

[17] S.J. Dickinson, A.P. Pentland, and A. Rosenfeld, "3d shape recovery using distributed aspect matching," *IEEE Trans. PAMI*, vol. 14, no. 2, pp. 174–198, February 1992.

[18] S.J. Dickinson, A.P. Pentland, and A. Rosenfeld, "From volumes to views: an approach to object recognition," *CVGIP: Image Understanding*, vol. 55, no. 2, pp. 130–154, 1992.

[19] Randal C. Nelson and Andrea Selinger, "Large-scale tests of a keyed, appearance-based 3-d object recognition system," *Vision Research*, vol. 38, no. 15, pp. 2469–88, 1998.

[20] C. Schmid and R. Mohr, "Local grayvalue invariants for image retrieval," *IEEE Trans. PAMI*, vol. 19, no. 5, pp. 530–534, 1997.

[21] Murase, Hiroshi, and Shree K. Nayar, "Visual learning and recognition of 3-d objects from appearance," *International Journal of Computer Vision*, vol. 14, no. 1, pp. 5–24, 1995.

[22] Schiele, Bernt, and James L. Crowley, "Object recognition using multidimensional receptive field histograms," in *Fourth European Conference on Computer Vision*, Cambridge, UK, 1996, pp. 610–619.

[23] D.G. Lowe, "Similarity metric learning for a variable-kernel classifier," *Neural Computation*, vol. 7, no. 1, pp. 72–85, 1995.

[24] J.H. Friedman, J.L. Bentley, and R.A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Software*, vol. 3, pp. 209–226, 1977.

[25] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, Massachusetts, 1990.

[26] R.F. Sproull, "Refinements to nearest-neighbor searching in k-dimensional trees," *Algorithmica*, vol. 6, pp. 579–589, 1991.

[27] J. Ben-Arie, "The probabilistic peaking effect of viewed angles and distances with application to 3d object recognition," *IEEE Trans. PAMI*, vol. 12, no. 8, pp. 760–774, August 1990.

[28] R. Malik and T. Whangbo, "Angle densities and recognition of 3d objects," *IEEE Trans. PAMI*, vol. 19, no. 1, pp. 52–57, January 1997.

[29] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.

[30] D.G. Lowe, "Fitting parametrized three-dimensional models to images," *IEEE Trans. PAMI*, vol. 13, no. 5, pp. 441–450, May 1991.

[31] J.E. Lloyd, J.S. Beis, D.K. Pai, and D.G. Lowe, "Model-based telerobotics with vision," in *Proceedings ICRA '97*, Albuquerque, New Mexico, April 1997, pp. 1297–1304.

[32] J.E. Lloyd and V. Hayward, "Multi-rccl user's guide," Tech. Rep. Technical Report, Center for Intelligent Machines, McGill University, April 1992.

[33] J. Canny, "A computational approach to edge detection," *IEEE Trans. PAMI*, vol. PAMI-8, no. 6, pp. 679–698, June 1986.