Paul G. Gottschalk
Jerry L. Turney
Trevor N. Mudge

Robotics Research Laboratory
University of Michigan
Ann Arbor, Michigan 48109

Efficient Recognition of Partially Visible Objects Using a Logarithmic Complexity Matching Technique

Abstract

An important task in computer vision is the recognition of partially visible two-dimensional objects in a gray scale image. Recent works addressing this problem have attempted to match spatially local features from the image to features generated by models of the objects. However, many algorithms are considerably less efficient than they might be, typically being O(IN) or worse, where I is the number of features in the image and N is the number of features in the model set. This is invariably due to the feature-matching portion of the algorithm. In this paper we discuss an algorithm that significantly improves the efficiency of feature matching. In addition, we show experimentally that our recognition algorithm is accurate and robust. Our algorithm uses the local shape of contour segments near critical points, represented in slope angle-arclength space (θ-s space), as fundamental feature vectors. These feature vectors are further processed by projecting them onto a subspace in θ-s space that is obtained by applying the Karhunen-Loève expansion to all such features in the set of models, yielding the final feature vectors. This allows the data needed to store the features to be reduced, while retaining nearly all information important for recognition. The heart of the algorithm is a technique for performing matching between the observed image features and the precomputed model features, which reduces the runtime complexity from O(IN) to O(I log I + I log N), where I and N are as above. The matching is performed using a tree data structure, called a kD tree, which enables multidimensional searches to be performed in O(log) time.

The International Journal of Robotics Research,
Vol. 8, No. 6, December 1989,

1. Introduction

A problem that has received considerable attention in the computer vision literature is that of recognizing two-dimensional (2D) partially visible objects in a gray scale image. In addition to being an important problem whose solution has many practical applications, it is an important step toward the solution of the more difficult problem of recognizing three-dimensional (3D) partially visible objects in an image. The problem of recognizing partially visible objects is sometimes called the *bin of parts problem* because, in industry, parts are often presented for batch assembly piled in a bin. The general bin of parts problem (with no constraints on the objects that may appear in scenes except that they be rigid) has been described as the most difficult problem in automatic assembly (Mattill 1976). In this paper, we present a solution to the bin of parts problem where the objects are 2D or have a small number of distinct viewpoints that may each be treated as 2D objects.

There are three very general goals that should be common to all object recognition systems: accuracy, robustness, and efficiency. Accuracy is the ability of a recognition system to correctly segment an image into instances of the objects of interest, with as few false instances reported as possible. In addition, accuracy includes the reporting of the position and orientation of the objects of interest (henceforth referred to as the *pose* of the objects) with as little error as possible. Robustness is the ability of the recognition system to tolerate variations in the conditions in which it operates. Common degradations from the ideal include

image noise, sampled images, and lighting variations. We have attempted to provide enough experimental evidence to suggest that our algorithm is accurate as well as robust. In Section 5 we discuss the results of running our algorithm on a total of 20 images that were composed of two very different sets of parts. The recognition algorithm exhibited high accuracy on objects having more than about half of their boundaries exposed and gradually degraded as objects became more completely hidden.

A recognition system can be both robust and accurate, yet still not be practical. To make a system practical, it must also be suitably efficient. The efficiency of an object recognition algorithm can be gauged in terms of an order analysis or in terms of a set of time benchmarks. Employing both is preferable; the first to ascertain performance as the set of objects to be recognized is increased, and the second as a means of comparison with other techniques. We have adopted the view that low-level image operations such as edge detection, thresholding, filtering, etc., should not be included in the efficiency figures since the efficiency of these operations is highly variable, depending largely on what hardware is available. Additionally, the efficiency of off-line computations such as setting up a database of models or training is much less important than the efficiency of the on-line algorithm and so should be given separately.

This paper presents a 2D partially visible object recognition algorithm that is a development of ideas first outlined in Gottschalk, Turney, and Mudge (1987). In addition, we will attempt to characterize the accuracy, robustness, and efficiency of our approach to a greater extent than earlier work.

2. Related Previous Work

An object recognition algorithm may be classified according to two general attributes: the *features* that it uses and the *matching strategy* that it employs. We will not attempt to classify all the algorithms known to us; rather, we will examine those algorithms that are most closely related to our own. For a thorough review of much of the work relating to 2D object recognition, the reader is referred to Chin and Dyer (1986), Turney (1986) and Knoll and Jain (1986). Other important references that, while not closely related to our work, also address the topic of 2D object recognition include Fu (1974); Pavlidis (1977); Blum and Nagel (1978); Tropf (1980); Ballard (1981); Segen (1983); Ballard and Sabbah (1983); Bhanu and Faugeraus (1984); Cowan, Chelberg, and Lim (1984); Koch and Kashyap (1985); Ayache and Faugeraus (1986); and Dubois and Glanz (1986).

As will be discussed more later, our algorithm employs data-compressed vectors of samples from the slope-angle versus arclength (θ-s) representation of the edge contours of objects that are near to high-curvature points of the contour (critical points). Other workers who have used the θ-s representation of edges are Perkins (1978); Yam, Martin, and Aggarwal (1980); McKee and Aggarwal (1977); Turney (1986); and Tsui and Chan (1987). Freeman (1977) has used critical points as features (critical points are discussed in detail later).

It is often true that algorithms that use informative features (in the sense that a given model feature matches few other model features) will be able to perform matching more quickly than those that do not. There are two reasons for this: first, there will be fewer highly informative features than less informative features, and second, informative features usually provide a large vector of attributes that can be used to quickly reject those ⟨image-feature ↔ model-feature⟩ pairings leading to a faulty hypothesis. These two properties usually allow such systems to reduce computation. A disadvantage of such systems is that, due to the scarcity of informative features, if the system is designed to handle overlapping or partially visible objects, then the degree of occlusion that can be tolerated by the system is reduced. Bolles and Cain (1984) have used highly informative features to advantage. In their method, the features are comprised of a *focus feature* and a number of satellite features. The resulting composite features are very informative and rare, allowing Bolles and Caine to use an NP-complete matching procedure. Turney (1986) has also used highly informative features. His recognition procedure, however, remains robust to large degrees of occlusion in the images, since it falls back on less informative features if the most informative ones are not present. Recently, Chien and

Aggarwal (1987) detailed a method for recognizing 3D objects with features comprised of configurations of four high-curvature points from the silhouette boundary of the object. The informativeness of high curvature points is a likely contributor to its success. Lowe (1987) uses informative configurations of straight line segments, which he calls *perceptual groupings,* as features in the initial stage of matching in his algorithm.

While success of a recognition algorithm depends on the choice of good features, the design of the matching algorithm is even more critical. The matching process has been formulated as a subgraph matching problem (Bolles and Cain 1984; Cowan, Chelberg, and Lim 1984), as a tree search (Grimson and Lozano-Pérez 1987; Ayache and Faugeraus 1986; Goad 1983; Tropf 1980), as a Hough Transform (Turney 1986; Turney, Mudge, and Volz 1985; Knoll and Jain 1986; Schwartz and Sharir 1986), and as a parsing problem (Fu 1974). Our matching strategy does not fit neatly into any one of these categories, as it has elements of both tree searching and correlation over edge contours.

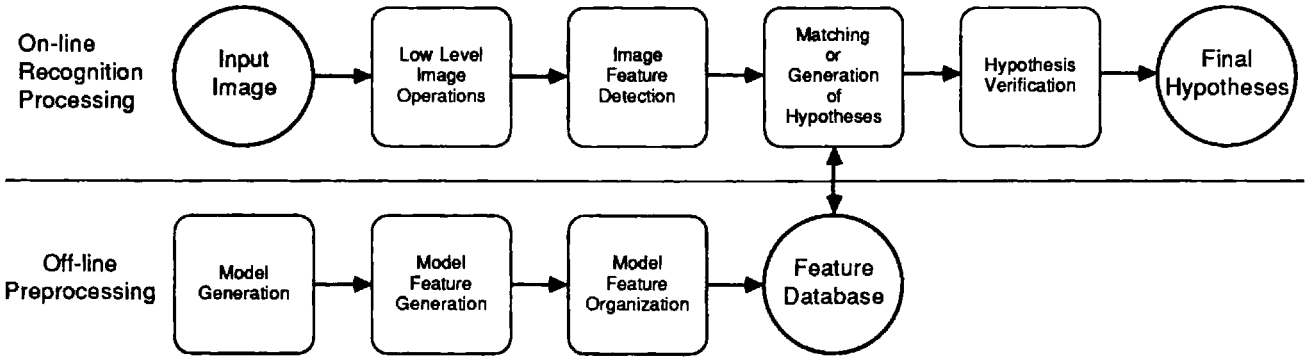
One of the major concerns in the design of our matching recognition algorithm, aside from the commonly considered ones of accuracy and robustness, is efficiency. Goad (1983) discusses an object recognition algorithm for 3D objects in which he attempts to speed up recognition of objects by pre-compiling a portion of the search tree and predetermining a best search path for a given object. We designed our matching algorithm in the same spirit as Goad by taking advantage of the freedom to perform off-line precomputation to optimize our algorithm's recognition efficiency.

Another method that is similar to ours is that of Schwartz and Sharir (1986). Like us, they have attempted to construct an efficient data structure for performing matching. The technique is called geometric hashing. In it they "hash" 5D feature vectors that have been obtained from the object models. Roughly speaking, their features are obtained by projecting constant-length windowed sections of a θ-s representation of the model boundaries onto the first four Fourier basis elements. The final feature vector is obtained by catenating to these four elements a fifth element representing the total angular change from one end of the window to the other. The "hashing" is accomplished by covering the entire 5D feature space with uniform hypercubes and using a hash function that maps a feature vector in a hypercube to that hypercube's bucket in the hash table. The term "hashing" is somewhat misleading in that it implies that the retrieval time is constant in the number of model features that are stored in the table. In fact, however, only the *entire set* of model features stored in a hypercube can be accessed in constant time, so, strictly speaking, the access time is linear in the average number of features in a hypercube. Therefore, when there are a large number of model objects, the size of the hypercubes must be reduced in order to maintain the fast access time. Unfortunately, shrinking the hypercubes has two undesirable effects: it increases the amount of storage necessary for the table, and, more seriously, it makes it less likely that a noisy image feature will "hash" into the hypercube with the correct model feature. Schwartz and Sharir attempt to get around this problem by examining the 32 neighboring hypercubes in 5D space of the hypercube that the image feature maps to. Under less favorable conditions than those reported, where high contrast parts and back-lighting were used, it is possible that this may not be sufficient. The essential problem with using Schwartz and Sharir's geometric hash table data structure is that it is difficult to query for the features in an arbitrarily sized and positioned hypercube in the 5D feature space while maintaining favorable access complexity. In our work, we have chosen a data structure that avoids the problems inherent in geometric hashing while maintaining logarithmic average access time. This data structure, called a kD tree, will be discussed in detail later. Geometric hashing is, nevertheless, an important step in the right direction, one that has lately been employed in work to recognize 3D objects (Lamdan, Schwartz, and Wolfson 1988).

3. Considerations for the Design of a Highly Efficient 2D Object Recognition Algorithm

Conceptually, the simplest strategy for recognizing 2D objects in an image is to attempt to match the model of each possible object at every position and orientation in the image. In two dimensions, this approach is

Fig. 1. Steps in the recognition algorithm.



computationally feasible, though very slow. In order to speed up the recognition procedure, most object recognition techniques use features of the objects. Typically the recognition procedure attempts to match the features from the models of the objects to features in the image. A feature is a processed version of the image data that has the desirable property of greatly reducing the data needed to represent the image adequately. At the same time, extracting a feature should sacrifice as little as possible of the information in the image necessary for recognition. Clearly, for feature-based recognition to be worthwhile, the time taken to extract the features from the image should be more than made up for by the reduced time needed to perform recognition using the features. Generally, the extraction of features is a relatively low-level operation and can often be done very quickly using special hardware. In the case where the recognition of partially visible objects is required, it is necessary that spatially local, or combinations of spatially local, features be used. Global features are too prone to distortion if an object in the scene is occluded by another.

Figure 1 shows the general strategy employed by our algorithm, as well as many other recognition methods. Examining the on-line half of Figure 1, we see that the input image is processed to extract higher level representations such as boundary segments, regions, or (as in our case) edge contours. These representations may then be further processed to detect the features. The features may be represented by an ordered set of numbers, often called a feature vector. The image feature vectors are then compared to all of the model feature vectors. Those model feature vectors that are close enough to an image feature vector, according to some metric, are hypothesized to exist in the image at the position and orientation given by the image feature vector.[1] These initial hypotheses must then be verified or rejected. The hypotheses that pass the final verification phase are those that are most likely to hypothesize the correct object appearing at the correct pose in the image.

The off-line preprocessing branch of Fig. 1 starts with a model generation phase. Models are typically generated by processing a set of training images or by a CAD system (Turney, Mudge, and Volz 1985). Features are then extracted from the models in the same way as in the feature detection phase in the recognition branch of the figure. As a final key step in off-line processing, the model features are organized into a data structure that is accessed during the feature-matching phase of the recognition procedure.

The relationship between the off-line step of model feature organization and the on-line step of feature matching deserves closer scrutiny. A very powerful means of speeding up the feature-matching stage is available through the creation of an appropriate data structure for the model features. This data structure should be designed to allow the fastest possible retrieval of the matching model features. Some algo-

1. Formally, a hypothesis is an ordered pair $(\mathcal{M}, \mathcal{T})$ where $\mathcal{M}$ is a model that is hypothesized to appear in the image, and $\mathcal{T}$ is a transformation that is applied to the model in order to place it at the hypothesized pose. In order to facilitate the discussion, we will sometimes speak of a hypothesis to be the model $\mathcal{M}$ after applying $\mathcal{T}$ to it.

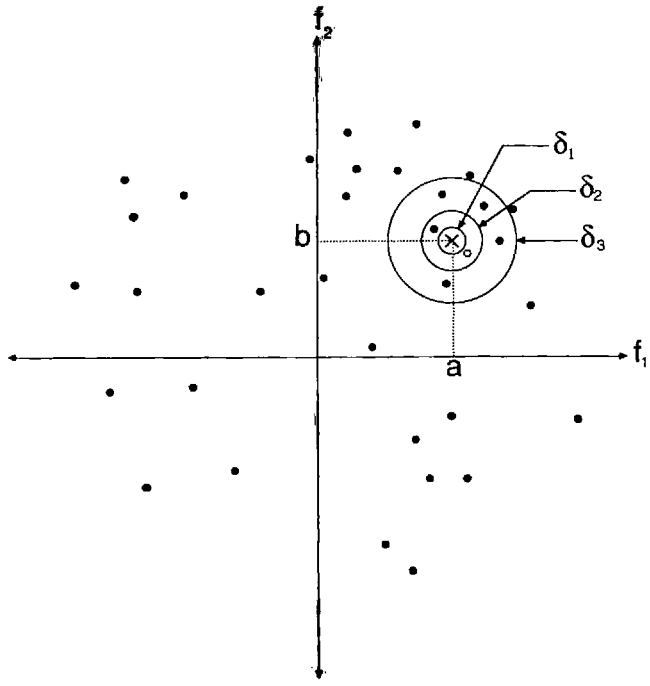
rithms have taken a step in this direction. For example, Turney, Mudge, and Volz (1986) sort the set of model features by their saliency (saliency formalizes the notion of informativeness of 2D features). However, the matching process is still a linear search. Knoll and Jain (1986) choose a set of features from the model so as to reduce overall recognition time. However, the improvement that this strategy can yield is limited since, again, the matching process is a linear search.

It is possible to organize the set of model features so that the feature matching is much more efficient than a linear search. To this end, it is useful to view the operation of retrieving those model feature vectors that match a feature vector from the image as an operation on an abstract data type: given a K-dimensional vector, **v**, return a list of all K-dimensional vectors from a set of vectors that lie within a K-dimensional neighborhood (as defined by some distance metric) of **v**. A number of data structures have been proposed in the database literature that allow this retrieval to be done in average case $O(\log N)$ time, where N is the cardinality of the set of vectors to be searched (Bentley and Friedman 1979). This is in contrast with the $O(N)$ time of a linear search. Figure 2 illustrates this view of the matching process. In Sec. 4, we discuss how one of these data structures, a kD tree, can be used to do very fast feature matching.

Further consideration of Fig. 1 yields some interesting insights into the object recognition process. One such insight concerns the selection of features. A degree of freedom that exists in the design of an object recognition algorithm is the choice of features. They should be chosen to be as informative as possible within the constraint that a certain degree of possible partial occlusion be allowed. As features become more informative, fewer of them will be found. If there are too few features, the chance of them being obscured increases and performance suffers. For example, the local feature focus technique of Bolles and Cain (1984) falls prey to this difficulty; the features are highly informative, but there are often very few of them, and so we would not expect this technique to perform well when high degrees of occlusion occur. Indeed, this expectation is borne out by experiments in Turney (1986).

Another observation that can be made from Fig. 1 is that the efficiency of the recognition process, as we have defined it, is the sum of the execution times of the last three stages: (1) feature detection in the image; (2) feature matching or hypothesis generation; and (3) hypothesis verification. There appears to be a fundamental trade-off between these three stages. However, to our knowledge, only Knoll and Jain (1986) have examined this trade-off to any extent. They assumed a two-stage model of the recognition process, in which the feature detection stage and the feature matching stage in Fig. 1 are treated as a single stage. Further, the features they used were assumed to be selected from the set of all boundary segments sampled at uniform intervals of arclength in the models. Under these conditions, they showed how to choose the features from the set of all of the boundary segments so as to reduce the total recognition time.

Fig. 2. A hypothetical 2D feature space. The solid dots represent model features, while the × represents the image feature located at the point (a, b) in the feature space. The hollow dot represents the model feature that correctly matches the image feature. The large, medium, and small circles depict Euclidean neighborhoods of the image feature with radii δ_1, δ_2, and δ_3 respectively.



4. Two-Dimensional Object Recognition Algorithm

In this section, we discuss our approach to 2D object recognition. The heart of the algorithm, the feature

matching technique, will be discussed first. The choice of features is an important consideration in the design of any object recognition algorithm. If done correctly, a synergy can be developed between the feature matching algorithm and the features that will significantly enhance the efficiency of the matching algorithm. A discussion of the features we use and their impact on the matching strategy follows the discussion of matching. Next we describe our hypothesis verification method. To conclude this section, we explain the way in which the overall algorithm combines the three central modules and derive the complexity of the algorithm.

4.1. Feature Matching

In our algorithm, as in a number of others, features of models of those objects that we desire to find in an image are stored in a database. As will be discussed in the next section, each feature employed by our algorithm encodes the shape of an object in the locality of a critical point or point of high curvature. The features in our system are represented by vectors that may be of any dimension, but in our experiments were five-dimensional. Our matching technique is very general and is completely independent of how the features are computed. The goal of the matching module is to retrieve, as quickly as possible, all features in the model-feature database that are close, in some sense, to a given image feature. Closeness is defined in terms of a metric over the feature space. Each feature in the set of model features that is returned as the result of such a query forms a conjecture, or hypothesis, as to which object in what pose resulted in a given feature in the image.

We emphasize the importance of an efficient matching algorithm: in our experimental system, there were only a few hundred features in the model-feature database. However, it is not difficult to imagine a practical system in which many thousands of features must be stored. Because the matching process must typically be performed on a significant fraction of the features in the image, efficiency becomes critical to the practicality of an algorithm. It is not uncommon to find recognition algorithms that perform a linear search through the database of model features to find the set of acceptable matches. Clearly this limits the practicality of such algorithms. Below we describe an $O(\log N)$ method for the search procedure.

To be more precise, the matching algorithm performs the following function: given a set of K-dimensional vectors, retrieve all those vectors in the set that fall within a K-dimensional δ neighborhood of another vector. We will henceforth call this operation a neighborhood search. As we mentioned, Fig. 2 illustrates this formulation of the matching problem. The definition of neighborhood includes the choice of a distance metric. One general class of metrics is defined by

$$D_n(\mathbf{a}, \mathbf{b}) = \left[\sum_{i=1}^{K} |a_i - b_i|^n \right]^{1/n} \tag{1}$$

where K is the dimensionality of the vectors $\mathbf{a}$ and $\mathbf{b}$, a_i and b_i are the components of $\mathbf{a}$ and $\mathbf{b}$ respectively, and n is the order of the metric. A special case of (1), the D_∞ metric, or Chebeychev metric, is obtained as $n \rightarrow \infty$ and can also be written as

$$D_\infty(\mathbf{a}, \mathbf{b}) = \max_i |a_i - b_i|. \tag{2}$$

A δ neighborhood of a vector $\mathbf{s}$ with respect to D_n, denoted $N(\delta, \mathbf{s}, n)$, is defined as the set of all vectors $\mathbf{r}$ such that $D_n(\mathbf{r}, \mathbf{s}) < \delta$. Furthermore, for later use, we note the fact that $D_\infty(\mathbf{a}, \mathbf{b}) \leq D_n(\mathbf{a}, \mathbf{b})$, $\forall\, \mathbf{a}, \mathbf{b} \in \mathcal{R}^K$ implies that $N(\delta, \mathbf{s}, \infty)$ contains $N(\delta, \mathbf{s}, n)$ for all finite n.

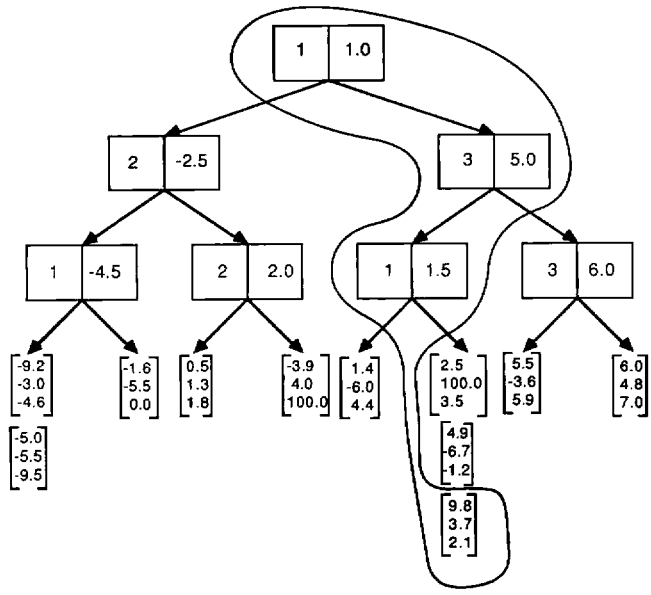
A neighborhood search using the Chebeychev metric can be reduced to a special case of a problem in database theory, namely that of multikey range searching. The problem of range searching can be stated as follows: given a set of records with K real valued keys, retrieve those records where the keys of all of the retrieved records fall within a range specified for each key. We can use range searching to retrieve all vectors of a set that are contained in $N(\delta, \mathbf{s}, \infty)$ by letting each component of the vectors be a key and choosing the ranges for each key to be the intervals $[s_i - \delta, s_i + \delta]$, $(i = 1, \ldots, K)$, where the s_i are the components of $\mathbf{s}$.

An important parameter in the neighborhood search is δ, because it controls the number of hypotheses that

will be generated, on average, for each image feature. For example, assume that an image feature is a noisy version of a certain model feature. If δ is made too small, the noise will make the probability that the model feature is within the neighborhood small, and the algorithm will often fail to find the correct object (see neighborhood δ_1 in Fig. 2). On the other hand, if δ is made excessively large, then the correct hypothesis will likely be among those retrieved by the neighborhood query (since the large neighborhood implied by a large δ will enclose most of the probability density of the image feature vector in the feature space). Unfortunately, as is evident in Fig. 2, a large neighborhood, δ_3, will likely include many incorrect hypotheses as well as the correct one. Since all hypotheses must be verified, and verification is computationally rather expensive, allowing δ to be indiscriminately large is not acceptable. We have chosen to view δ as a design parameter. The recognition task will require a certain level of reliability, and δ should be made just large enough that the correct hypothesis will be retrieved with a high enough probability to satisfy the reliability requirement. The ideal value, δ_2, can only be determined by experiment; however, our experience indicates that the number of hypotheses is rather insensitive to the size of δ over a fairly wide range, so long as it exceeds a minimum size.

There are a number of data structures that may be used to perform the range searching operation (Bentley and Friedman 1979). A data structure called a kD tree was chosen as being the most suited to our purpose: it has the best average case query time complexity, best preprocessing time complexity, requires minimum space, and it is the easiest to implement.[2] Figure 3 shows a kD tree for retrieving 3D vectors. A kD tree is a binary tree with two pieces of additional information stored at each node, a key identifier and a discrimination value. In our work, we are concerned with retrieving vectors and their associated records; consequently the key identifiers correspond to vector indices (i.e., the keys are the elements of the vector). The tree is organized such that at each node all data stored in the left subtree has the key indicated by the node's key identifier less than the discrimination value, while the data on the right has the key indicated by the node's key identifier greater than the discrimination value stored at the node. All data in a kD tree are stored in its leaves. The leaves are lists of less than some predetermined length that contain data satisfying the constraints of all of the ancestor nodes.

Fig. 3. A simple kD tree for rapid retrieval of 3D vectors. The enclosed area shows the sequence of nodes traversed to retrieve the one vector in the tree that is in the neighborhood $(10.0, 4.0, 2.0)^T + (\pm 0.5, \pm 0.5, \pm 0.5)^T$.



In our application, the kD tree need not support random insertions and deletions, i.e., all of the data in the tree is known a priori. Under these conditions, it is possible to balance the kD tree during its construction unless the data is highly degenerate.[3] This is done by first computing the variance of each key over all records. The key with the largest variance is selected as the root node's discrimination key (as will be described in greater detail later, organizing the tree in this way makes queries more efficient). Denote this key as α. The median of α is found and this value becomes the root node's discrimination value. The records are then divided into sets: those where α is less than or equal to the discrimination value (these records are stored in the left subtree), and those where α is greater

2. The "kD" in the name of the "kD tree" stands for "k-dimensional."

3. The degeneracy occurs when two or more of the keys of two distinct records have the same value.

than the discrimination value (these records are stored in the right subtree). The entire procedure is repeated recursively on each set to create the children of the root node. The recursion stops when there are less than a given number, say J (six in our case), records left in the set. J is chosen such that the cost of a query is minimized. In the case where there are less than J records left in the set, a leaf, which is a linked list of less than J elements, is formed.

Searching a kD tree is a simple recursive procedure. A range is specified for each key; the task is to retrieve all records where the value of every key falls into the corresponding range for that key. At each node that is not a leaf (starting at the root), the range associated with the discrimination key of the node is compared with the discrimination value stored at the node. If the range interval lies completely above the discrimination value, the result of a recursive call made on the right subtree only is returned. Similarly, if the range interval lies completely below the discrimination value, the result of a recursive call made on the left subtree only is returned. If the range interval straddles the discrimination value, recursive calls on both the left and right subtrees are made. The results of the calls are concatenated and then returned. The recursion stops when a leaf is encountered, and the list at the leaf is scanned; any records whose vector of keys do not fall in all of the range intervals are excluded. A linked list of the remaining records is returned.

We now return to the construction of the kD tree. As was mentioned previously, during the construction of a kD tree, the discrimination key of a node is chosen as the key that has the greatest variance over the set of remaining records. It was also mentioned that this was done to make queries more efficient. Having described how queries work, we may now understand why this is so. The query algorithm makes two recursive calls if the range associated with the discrimination key at a given node contains the discrimination value. If the kD tree can be constructed such that this case occurs less frequently than the other two cases (where the range lies completely above or completely below the discrimination value), then the query will be more efficient, since only one recursive call will be made. This is especially important in the nodes near the root, since avoiding a search of both sides of the tree near the root yields the greatest savings. This is why the key with the greatest variance is made the discriminator of the root of the remainder of the tree. If we assume that the ranges are distributed like the data (which they are in our case), it is less likely that a range will contain the discrimination value if that key has a large variance.

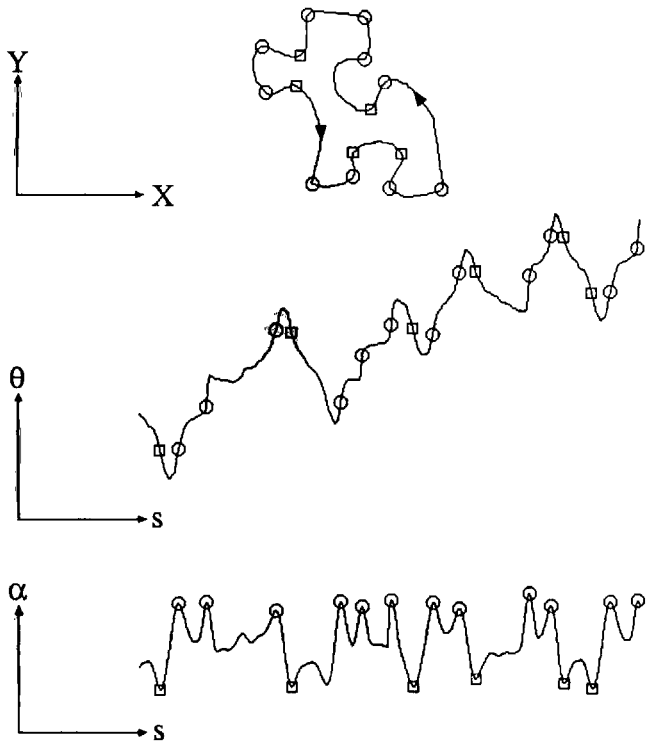
We have shown that the kD tree can perform neighborhood searches with respect to the D_∞ metric. It is simple to modify the algorithm to perform neighborhood searches with respect to all of the metrics represented by (1) with no increase in complexity for either queries, space, or preprocessing. It was noted previously that for a given δ, $N(\delta, \mathbf{s}, \infty)$ contains $N(\delta, \mathbf{s}, n)$. Thus, all that must be done to implement a neighborhood search for finite n is to scan the result of a $N(\delta, \mathbf{s}, \infty)$ search implemented by a K-dimensional range search using a kD tree and exclude those vectors that do not fall into $N(\delta, \mathbf{s}, n)$. We can show that the time complexity of a query remains the same as in the $N(\delta, \mathbf{s}, \infty)$ case as follows. It is shown in Bentley and Friedman (1979) that the average complexity of a query is $O(\log N + F)$, where N is the size of the set of vectors to be searched, and F is the size of the set of vectors that is returned. The complexity of scanning the returned set and rejecting the vectors that are not in the neighborhood $N(\delta, \mathbf{s}, n)$ is $O(F)$. Therefore the complexity of a general neighborhood search remains $O(\log N + F)$.

In this section we have discussed the problem of feature matching, and we have shown that the feature matching problem is isomorphic to the problem of neighborhood searching. We have also shown how to implement neighborhood searches, and hence feature matching, quickly via kD trees. In the following section, we discuss our approach to hypothesis verification, and, in particular, we show how kD trees may also be used to some advantage there.

4.2. Selection and Computation of Feature Vectors

Our algorithm recognizes objects entirely by the shape of contours. A contour consists of edge points that have been linked together into a single edge segment and stored in a data structure, typically a linked list.

Fig. 4. The puzzle piece contour represented in both Cartesian space (top) and θ-s space (middle). At the bottom is shown the α(s) curve. Marked on all three curves are the locations of critical points. Positive extrema of curvature are marked with circles, and negative extrema of curvature are marked with squares.



Contours are represented in a dual Cartesian and slope angle-arclength (*θ-s* hereafter) representation. The ordinate, or θ, in the *θ-s* representation is the slope angle of the tangent at the point of interest on the contour, measured relative to the horizontal. Similarly, the abscissa, or s, is the arclength measured from some arbitrary point of reference on the contour to the point of interest. As an illustration of the *θ-s* representation, Figure 4 shows a contour of a jigsaw puzzle piece represented in Cartesian space and in *θ-s* space.

We have a number of reasons for choosing to represent contours in *θ-s* space. One fundamental consideration is that the *θ-s* representation simplifies the construction of features that are invariant to image translation and rotation. Translation invariance is automatic, since all quantities are measured from a point of reference on the contour. As for rotation invariance, note that the rotation of a contour in Cartesian space corresponds to a simple shift in the ordinate (θ) of the *θ-s* representation. To normalize contours with respect to rotation, an offset is added to θ so that the reference point on the contour is some standard value, such as zero (in the following paragraphs, we will describe how we use points of high curvature, or *critical points* as the reference points). If this is done, then the *θ-s* representations of rotated versions of the same Cartesian contour are the identical. Other advantages of the *θ-s* representation, such as single-valuedness and the ease with which our features can be extracted, contribute to the efficiency of the algorithm at various stages. These aspects of the *θ-s* representation will be explained in more detail as the pertinent portions of the algorithm are discussed.

The computation of the *θ-s* representation for contours is only an intermediate step to the extraction of the final feature vectors in our algorithm. Since we have as a goal the recognition of partially visible objects, we must employ spatially local or a combination of spatially local features. We have chosen our fundamental geometric features to be neighborhoods of critical points (i.e., fixed length portions of contours with critical points at their centers). Attneave (1954) was one of the first to discuss the importance of points of locally maximal curvature, which he termed *critical points*. He also suggested that they have at least one aspect of a good feature, namely that they are highly informative in the sense that humans can often recognize an object given only the critical points on the object's line drawing. More recently, Biederman (1985) has presented evidence that critical point–like entities occurring at points of concavity along a boundary play a significant role in human vision. Attneave's and Biederman's conjectures receive empirical support in the domain of machine vision in Mudge, Turney, and Volz (1987).

As noted earlier, for the case of 2D object recognition, a formalized notion of the degree of informativeness of features, called saliency, was defined in Turney (1986). Under the condition that features are contour segments of a given length from an object set, Turney noted that the most salient of these features usually contains a critical point. Figure 5 shows a map of the saliency of segments making up the contours of a 2D object. Note that the most informative segments are highly structured (in the sense that they contain a large amount of high curvature boundary) and usually contain at least one critical point. This implies that, typically, the neighborhoods of critical points are also highly informative.

Fig. 5. A perspective view of the contour of a doorlock part. Extending from evenly spaced points on the contour are vertical bars whose lengths are proportional to the informativeness of the contour neighborhoods, which have the points as their centers. Informativeness in this case is the inverse of the number of times the contour segment of interest matched sufficiently well in the object set. Note the large correlation between the amount of high curvature boundary that is contained in the segment and how informative it is.

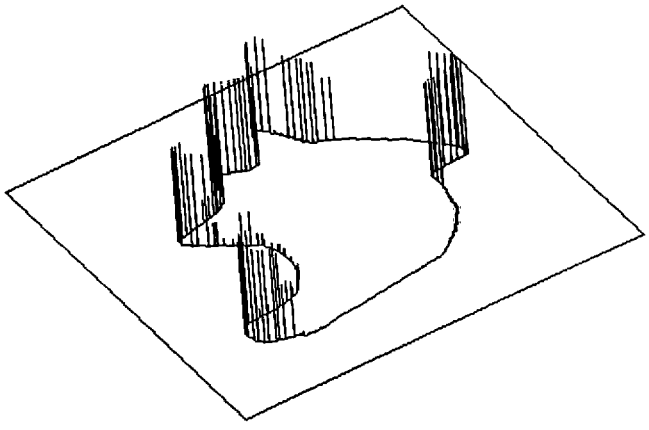
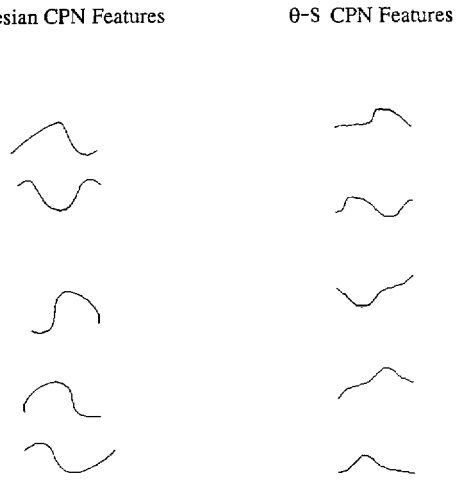


Fig. 6. On the left are several typical puzzle piece CPN features in their Cartesian representation, and on the right are the same features represented in θ-s space.



Critical points have another advantage in addition to their high information content: they are easily and quickly extracted by the application of a one-dimensional edge operator to the θ-s representation of the contours. This is clear from the following:

> A critical point is a point of locally maximal curvature, and curvature is the rate of change of slope with respect to arclength. Thus, given a contour represented in θ-s space, denoted $\theta(s)$, the curvature is $\alpha(s) = d\theta(s)/ds$ and the critical points are the peaks in this function. Application of a one-dimensional derivative of a Gaussian edge detector to the $\theta(s)$ function performs precisely the operation d/ds, in addition to filtering any noise that may be present.

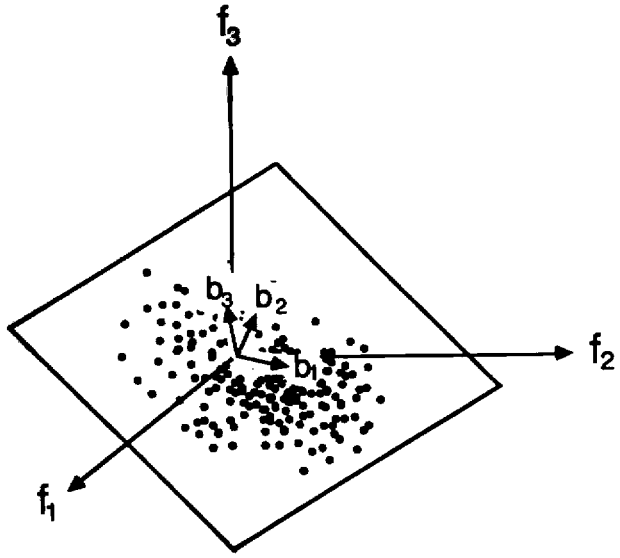
Figure 4 shows a contour whose critical points have been marked in its Cartesian representation as well as its θ-s representation. The critical points in the figure are the edge points marked by a one-dimensional derivative of a Gaussian edge detector.

Critical point neighborhood (CPN) features, while being much more informative on average than other segments of a set of contours, are nevertheless not a very efficient encoding of the important recognitive information. Examination of Figure 6 reveals that CPNs are continuous and, in fact, rather similar in appearance. As we have noted, in practice the representation is discretized and contains a finite number of samples that comprise a vector. The similarities in Figure 6 suggest that the elements of a CPN feature vector are highly correlated with each other, i.e., it is possible to predict quite accurately what the value of an element will be given the values of some other elements of the CPN feature vector. The Karhunen-Loève (K-L) expansion, a standard data compression technique in signal processing, takes advantage of highly correlated data. Rosenfeld and Kak (1976) describe how the K-L expansion can be employed with success to compress picture data. We use it in the present work to reduce the data needed to represent the CPN features described above.

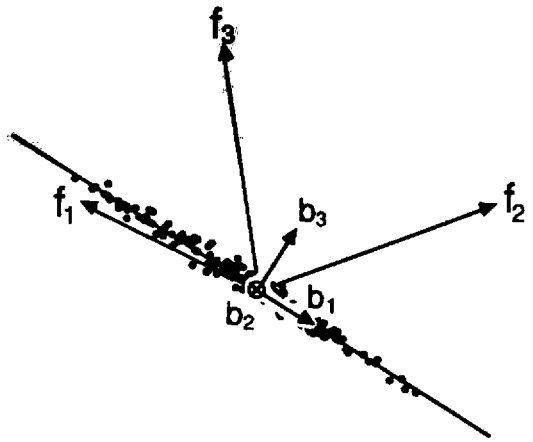
Each sample of a CPN feature vector can be considered to be a component of a real vector of some dimension, say N. The set of CPN features can now be viewed as the result of trials of an underlying real random vector $\mathbf{X}$ of dimension N. We may assume that $\mathbf{X}$ is zero mean since, if it is not, the mean may be estimated and $\mathbf{X}$ adjusted accordingly. Let $\mathbf{R} = E[\mathbf{X}\,\mathbf{X}']$ be the auto-correlation matrix of $\mathbf{X}$. Since $\mathbf{R}$ is non-negative definite, there exists a set of orthonormal eigenvectors and associated eigenvalues of $\mathbf{R}$, ϕ_k and $\lambda_k \geq 0$ respectively, where $k = 1, \ldots, N$. Define the random variables $Y_k = \phi_k' \mathbf{X}$. Without loss of generality, we may assume that the eigenvectors ϕ_k are ordered so that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_N$. The K-L expansion says that $\mathbf{X}$ can be expanded in the following manner:

$$\mathbf{X} = \sum_{k=1}^{N} Y_k \phi_k, \tag{3}$$

Fig. 7. A hypothetical 3D feature space is shown in (A), whose elements are highly correlated among themselves (dots represent features in the space). The correlation can be seen from the degree to which the features cluster near the plane. Shown in (B) is a different view of (A) looking with $\mathbf{b}_2$ going into the paper. The vectors $\mathbf{b}_1$ and $\mathbf{b}_2$ span the plane, while $\mathbf{b}_3$ is orthogonal to it. When represented in terms of the basis $\mathbf{b}_1$, $\mathbf{b}_2$, and $\mathbf{b}_3$, the $\mathbf{b}_3$ component of the vectors will be near zero and may be ignored, achieving a degree of data reduction. The K-L expansion allows such a basis to be computed.



A



B

where the Y_k are uncorrelated and var $(Y_k) = \lambda_k$.

Geometrically, the K-L expansion chooses a special basis in the N-dimensional vector space in which $\mathbf{X}$ is defined. This basis has the following property: the basis vector ϕ_1 defines the direction in which $\mathbf{X}$ has the greatest variance (i.e., Y_1, the projection of $\mathbf{X}$ in the ϕ_1 direction, has the maximum variance); the second basis vector ϕ_2 defines the direction in the subspace perpendicular to ϕ_1 in which $\mathbf{X}$ has the greatest variance, and so on until all dimensions are defined. Therefore, the K-L expansion chooses a basis that, when $\mathbf{X}$ is represented in terms of it, will concentrate the total variance of $\mathbf{X}$ into its lower numbered components. A subspace whose basis vectors ϕ_k are associated with those Y_k having small variance may be ignored with negligible effect upon the stochastic properties of $\mathbf{X}$. The result of this is that the original feature vectors can be projected onto a space of smaller (often considerably smaller) dimension and still retain most of their information. Figure 7 gives an example of such a situation.

Before applying the K-L expansion to compress the data needed to represent the CPN feature, it is necessary to adjust the data to give it a zero mean to decorrelate the Y_k. Let $S = \{\mathbf{f}_i', i = 1, \ldots, V\}$ be the set of all feature vectors in the object set (extracted from models or training images), where V is the number of CPN features found in the training images. The set of zero mean feature vectors derived from S is $T = \{\mathbf{f}_i = \mathbf{f}_i' - m, i = 1, \ldots, V\}$, where $\mathbf{m}$ is the sample mean of S. The autocorrelation matrix, $\mathbf{R}$, can then be estimated from T by the formula

$$\mathbf{R} = 1/V \sum_{k=1}^{V} \mathbf{f}_k \mathbf{f}_k^t. \tag{4}$$

Using this estimate of $\mathbf{R}$, the values of ϕ_k and λ_k can be computed, and then the K-L expansion formula can be applied. The data reduction is effected by retaining only those basis vectors ϕ_k associated with the Y_k having the largest variances and then projecting the original features onto the subspace spanned by the retained ϕ_k. Define the total variance of $\mathbf{X}$, σ_x^2, as $E[\mathbf{X}^t\mathbf{X}]$, then $\sigma_x^2 = \Sigma_{k=1}^{N} \lambda_k$, since the ϕ_k are an orthonormal set. The number of ϕ_k's retained, L, is determined by the fraction of σ_x^2 we wish to retain.[4] The

4. The fraction is a design parameter. Adjusting the fraction allows data reduction to be traded off for informativeness. If the fraction is very close to 100%, there will be less data reduction, and the reduced features will represent the original features more closely. On the other hand, a lower fraction will increase the data reduction at the expense of a less perfect representation of the original features. As discussed in the text, a fraction quite close to 100% (e.g., 98%) still yields a large data reduction.

Fig. 8. On the left are the basis vectors represented in θ-s space. They result from applying the K-L expansion to all of the CPN features in the model set (in this case, a set of ten puzzle pieces). In the center is the percentage of the total variance associated with each basis vector. At right are the Cartesian representations of the basis vectors.

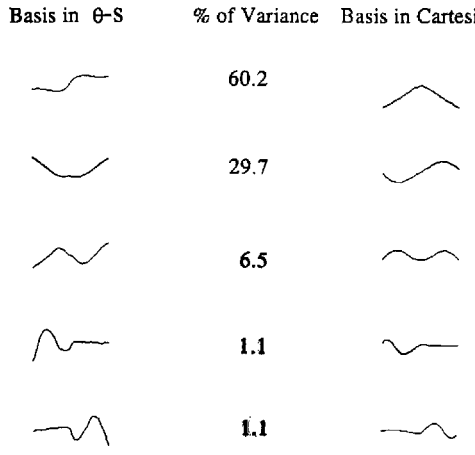
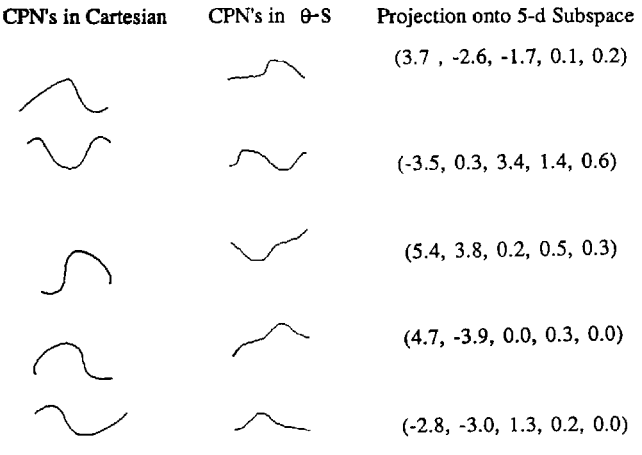


Fig. 9. On the left are shown some typical puzzle piece CPN features in Cartesian space. In the middle are their θ-s representations. On the right are their projections onto the reduced basis made up of the five vectors shown in Fig. 8.



reduced feature vector, **r**, of any CPN feature **f** can then be computed by the formula $\mathbf{r} = \mathbf{Pf}$, where **P** is the $L \times N$ matrix whose rows are the L retained ϕ_k ordered such that the ϕ_k with the largest associated variance appears at the top, the ϕ_k with the second largest variance appears second from the top, and so on to the bottom where the last retained ϕ_k (associated with the smallest variance) appears.

Figure 8 shows the K-L basis vectors, ϕ_k, which have been computed using all of the CPN features from the set of models of jigsaw puzzle pieces, the associated variances of the Y_k, and the reduced feature vectors. In our case, we required that 98% of the total variance be retained; only five dimensions out of a total of 45 were necessary to achieve the 98% variance figure. This is a reduction in data by nearly an order of magnitude. Finally, Figure 9 shows the projections of some typical CPN features onto the reduced basis obtained from applying the K-L expansion to the CPNs of a set of puzzle piece contours.

We have previously alluded to an important synergy that exists between the matching method and the feature computation technique. The kD tree matching would be significantly less efficient if the K-L expansion were not applied to the original feature vectors to yield the reduced feature vectors with uncorrelated, high-variance components. As discussed earlier, the logarithmic complexity of retrieval from a kD tree is only attained if no more than a small proportion of the nodes that are visited require both subtrees to be examined. If this requirement cannot be met, then the performance of a retrieval degrades to its worst case linear complexity. If the data is highly correlated, and a kD tree is built, it is much more likely that both subtrees will need to be searched during a retrieval than if uncorrelated, high variance data is used. Therefore, it is critical to the overall performance of the kD tree matching that it be used in conjunction with feature vectors that have been reduced using the K-L technique.

4.3. Hypothesis Verification

We now turn to the problem of hypothesis verification. In our framework, a hypothesis is a prediction about what may appear in the image, and verification is the process of comparing the prediction with some set of observations. Hypothesis verification consists of a detailed comparison of the model of the hypothesized object at the hypothesized pose with the image (or data derived from the image). The purpose of the comparison is to gather evidence, positive or negative, about the hypothesis in question. The result of the comparison is a score that rates the strength of the hypothesis. The score depends in some way on how closely what actually appears in the image matches what is hypothesized to appear. All the available hypotheses are rated, and those that have enough support are kept. These remaining hypotheses are the algo-

rithm's best guesses as to which objects appear in the image, and their poses.

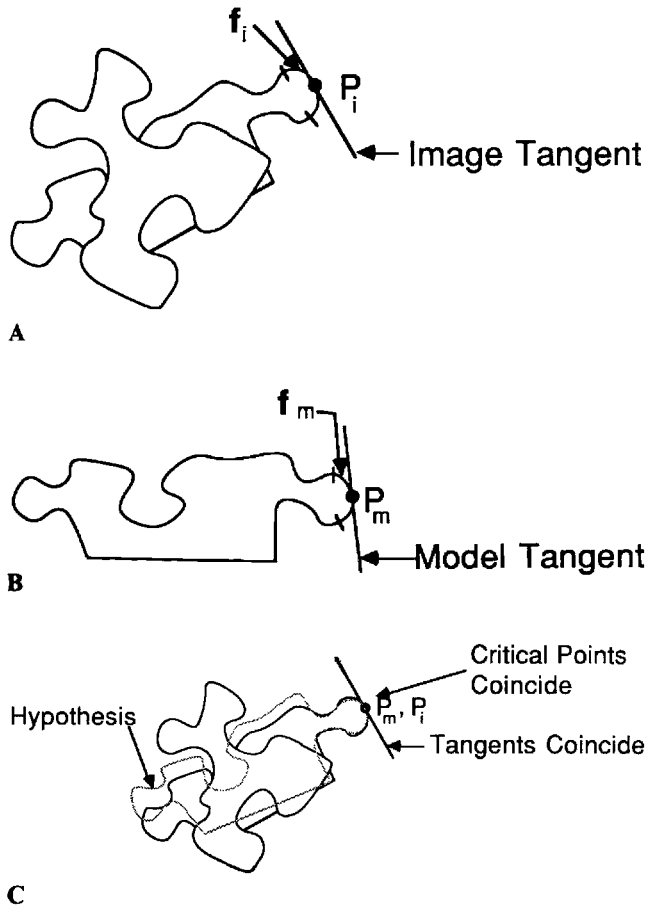
In terms of the discussion in Sec. 4.1, a hypothesis is a model that has a reduced CPN that falls within the δ-neighborhood of some image feature identified during the feature detection stage. The model, then, represents our guess at the specific object in the image that gave rise to the image feature.

The model of the object contains the reduced feature and its position within the model. More especially, the model of the object is the set of all of its Cartesian and θ-s contours, together with a list of the poses of all CPN features in the model. This information allows the model's pose to be transformed to the pose indicated by the feature found in the image (and therefore allows the list of critical points from the model to be transformed to their expected locations in the image). It will be convenient in the rest of the discussion to speak of a hypothesis as being the contours and critical points of the model after the pose transformation that aligns the model feature to the image feature (where the model feature matched the image feature), not just the information necessary to effect the transformation. Figure 10 illustrates the process of transforming the model's contours and critical points so that the matching model and the image features are brought into alignment.

Our verification scheme bases its decision about whether to pass or fail a hypothesis on two statistics: a fraction of critical points matched Q_{cp}, and a fraction of boundary matched Q_b. We chose these two statistics because past experience showed them to be effective decision variables for a wide variety of objects. In addition, other researchers have used similar measures successfully. For example, Bolles and Cain (1984) describe a measure that is similar to our computation of Q_b, and Ayache and Faugeras (1986) describe a measure that is similar to Q_{cp}, except that they use line segment features instead of critical points, as in this work.

Unlike other researchers, we have chosen to use two measures, as we have found that each possesses distinct strengths. In particular, Q_{cp} is effective for objects possessing many critical points; it heavily weights the most informative portions (the critical point neighborhoods) of an object's contour. However, the statistic Q_{cp} alone is not adequate for all kinds of objects. Some

Fig. 10. This figure illustrates how a model is transformed into a hypothesis using a feature that was matched during the feature matching stage of the algorithm. The image is shown at (A), the model at (B), and the image with the dotted outline of the hypothesis superimposed in (C). The reduced model CPN feature vector derived from $\mathbf{f}_m$, shown in (B), matches the reduced image CPN feature vector derived from $\mathbf{f}_i$, shown in (A). The critical points associated with these features are P_m and P_i, respectively. To create a hypothesis, P_m is translated so that it coincides with P_i and is rotated so that the tangent to the model contour at P_m is aligned with the tangent to the image contour at P_i. (C) shows the dotted outline of the hypothesis resulting from this process.



objects, such as nails, have relatively few critical points. In these cases employing Q_b in conjunction with Q_{cp} is appropriate, since these objects often have only three or four critical points visible; all the rest may be occluded by other objects. This is generally more than adequate for generating hypotheses, but it is inadvisable to rely solely on properties of the critical points for hypothesis verification. For example, Q_{cp} becomes very sensitive to accidental matches when there are few critical points on the object. On objects

with few critical points, all contour segments are roughly equally informative, hence the presence of any hypothesized contour in the image can be regarded as evidence that the hypothesized object was indeed present in the scene.

The idea of searching for features predicted by the model and using their presence to strengthen hypotheses is not new. As mentioned above, a quality measure very similar to Q_{cp}, with the difference that the features were line segments rather than critical points, can be found in Ayache and Faugeras (1986). In that work, however, the process is taken a step further by using a Kalman filter to recursively update a least-squares estimate of the model's position and orientation each time a new image line segment is found that matches closely to a model line segment. This step provides a more precise positional estimate than our method. It would be a simple matter to add this capability, if the application requires precise pose information. A simple least-squares fit would be sufficient, especially in light of the work in Turney (1986), where it is shown that the Kalman filter reduces to a least-squares fit when it is used for estimating the pose of motionless objects.

We now detail the computation of the statistics. The value of Q_{cp} is computed by searching a spatial neighborhood of each critical point from the hypothesis for a matching critical point in the image, and incrementing a count if one is found. In order to match, a critical point from the image must possess roughly the same orientation and the same sign of curvature as the hypothesized critical point it is being matched against. The orientation of a critical point is simply the value of θ of the contour at the critical point (recall that we consider the pose transformation to have already taken place). The value of Q_{cp} is then given by

$$Q_{cp} = I_{cp}/M_{cp}, \tag{5}$$

where I_{cp} is the number of critical points in the hypothesis that were found to match an image critical point, and M_{cp} is the total number of critical points in the model. In order to check quickly whether there are any critical points in the image matching a hypothesized critical point, a second kD tree is employed. For reasons that will shortly become apparent, we shall call this kD tree the *pose tree*. The pose tree is built during the feature detection stage when the CPNs are being found. The k^{th} CPN in the image is assigned a key vector $(x_k, y_k, \sin\theta_k, \cos\theta_k)$, which we shall call the pose vector. The elements of the pose vector are defined as follows: the ordered pair (x_k, y_k) is the coordinates of the k^{th} critical point in the image, and θ_k is the slope angle of the contour at the critical point. The sine and cosine of θ_k were used in place of θ_k itself to avoid branch discontinuity problems associated with direct representation of slope. Each hypothesized critical point is also assigned a pose vector. The pose tree is then used to perform a range query to retrieve all image critical points with roughly the same pose as the hypothesized critical point. Let the hypothesized critical point have the pose vector $(x_h, y_h, \sin\theta_h, \cos\theta_h)$. The range is then defined by the four-dimensional interval $(x_h \pm dx, y_h \pm dy, \sin\theta_h \pm d\sin\theta, \cos\theta_h \pm d\cos\theta)$. the values of dx, dy, $d\cos\theta$, and $d\sin\theta$ are not critical; they must be fairly large to allow for slight differences in pose of the hypothesis and an instance of the object in the image. We used 4 pixels for dx and dy, and 0.5 for $d\sin\theta$ and $d\cos\theta$. If the list returned by the range query is not empty, then the count I_{cp} is incremented. This process continues until all of the hypothesized critical points are checked, at which time Q_{cp} may be computed.

We have discussed the computation of the first statistic, which is based on the count of matched critical points. We now explain how we perform the computation of the second statistic, the fraction of boundary matched. The mechanics of performing the boundary comparison are quite straightforward. Figure 11 illustrates the process. The image contours are first drawn onto a bitmap to allow easy checking of the spatial proximity of contours (the contours are drawn white on black). After drawing the image contours, the contours of the hypothesis are traced, sample by sample, creating the Cartesian representation of the hypothesis. As before, the transformation from the model to the hypothesis is chosen so that the pose of the hypothesized CPN feature is at the same pose as the image CPN feature that it matched. At fixed intervals of arclength, ds, a probe is made along a line perpendicular to the hypothesis contour. The pixels on the probe line are generated by Bressenham's line algorithm (Bressenham 1965) such that the line probed is perpendicular to the hypothesis contour, and the pixels in the line

Fig. 11. The calculation of the percentage of boundary matched statistic, Q_b. Shown is an image contour (A) and a corresponding hypothesized contour (B). The hypothesis contour contains a CPN feature that matched an image CPN at (C). As explained in Fig. 10, the pose of the hypothesized part has been transformed so that the matched features have the same pose. In order to calculate the percentage of boundary matched, probes (D) are made perpendicular to the hypothesis contour in lines extending five pixels on either side of it. If a probe line intersects an image contour, a probe hit (E) has occurred, otherwise it is a miss (F). For each probe hit that occurs, Q_b (which is initialized to 0) is incremented by the percentage of the total hypothesis contour between two consecutive probe lines (G).

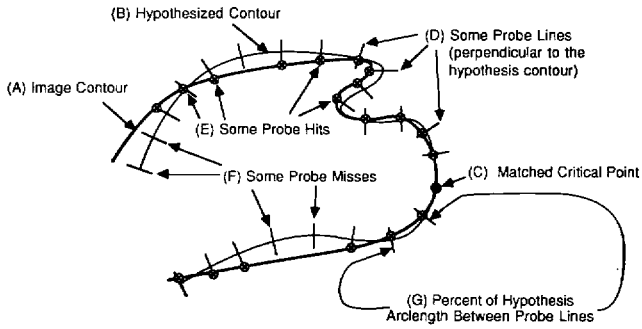
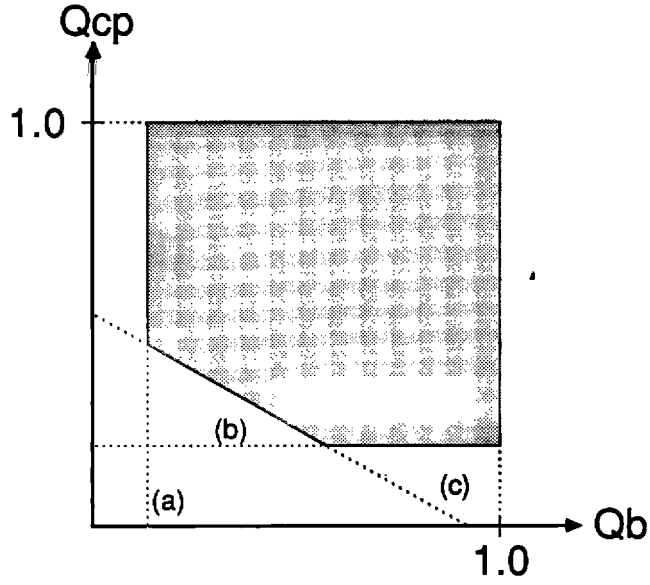


Fig. 12. In order to be accepted, a hypothesis must fall into the gray region. This region is the intersection of the half-planes defined by vertical line (a), horizontal line (b), and oblique line (c), and bounded above by 1.0 in both the Q_{cp} and Q_b directions. Four parameters, described in the text, determine the shape of the region. Optimal values of the parameters vary with the set of objects and must be determined experimentally. Performance is not sensitive to the precise values of the parameters.



are checked up to two pixels on either side of the contour. If a white pixel (which corresponds to a point on one of the image contours) is encountered in the probe, the fraction of boundary statistic, Q_b, is incremented by the quantity ds/s_t, where s_t is the total arclength of all the contours making up the hypothesis. The process continues until all of the contours making up the hypothesis have been probed.

We have discussed the methods used to compute Q_{cp} and Q_b, but we have yet to explain how these statistics are used to make the decision to accept or reject a hypothesis. As would be expected, the optimal decision regions depend on the object set as well as on the degree of occlusion allowed. While we do not find optimal decision regions, we nevertheless desire to be general enough to get good performance for a wide variety of object sets and degrees of occlusion. In particular, a hypothesis is accepted if the ordered pair (Q_{cp}, Q_b) falls into the following region:

$$\{(x, y): \quad x > T_1, y > T_2, \text{ and } \beta x + (1-\beta)y > T_3\}. \quad (6)$$

The hypothesis is rejected otherwise. The three thresholds and β are chosen to give the best performance with the given object set. Figure 12 shows a typical acceptance region.

4.4. Summary of the Algorithm and Complexity Analysis

The previous three sections have dealt in detail with the matching, feature detection, and hypothesis verification. In this section, we shall summarize the algorithm and analyze its complexity.

Let N be the number of model features, let b be the average number of critical points per unit arclength of contour in the model set, let P be the number of objects in the image, and let I be the number of features detected in the image. We shall examine the off-line computation first, ignoring the low-level operations of edge detection and edge linking. The off-line computation is composed of the following series of steps (which have been discussed in detail above): critical point detection, neighborhood extraction, K-L expansion, basis reduction, projection of model CPNs, and building of the kD tree. These steps have complexity $O(N)$, $O(N)$, $O(N)$, $O(1)$, $O(N)$, and $O(N \log N)$, respectively. Thus the entire procedure has complexity $O(N \log N)$.

The on-line portion of the algorithm is composed of two major parts: first a sequence of steps that are executed only once for each image that the algorithm is

asked to process, and a second sequence of operations that form a loop. We have written the on-line portion of the algorithm in Pascal-like pseudocode below:

```
Procedure ONLINE
BEGIN
        DETECT-CRITICAL-POINTS;
        EXTRACT-NEIGHBORHOODS;
        POSE-TREE-CONSTRUCTION;
        PROJECT-CPNS;
LOOP:   GET-NEXT-FEATURE;
        NEIGHBORHOOD-SEARCH;
        VERIFY-HYPOTHESES;
        REMOVE-ASSIGNED-
            FEATURES;
        IF MORE-FEATURES THEN
            LOOP ELSE DONE
END
```

The function of the first four steps should be clear from their names and the earlier discussion. The procedure DETECT-CRITICAL-POINTS has complexity $O(I/b) = O(I)$; the procedure EXTRACT-NEIGHBORHOODS has complexity $O(I)$; the procedure POSE-TREE-CONSTRUCTION has complexity $O(I \log I)$; and the procedure PROJECT-CPNS has complexity $O(I)$. Thus, the total complexity of the four steps prior to the loop is $O(I \log I)$. We now examine the loop body. The procedure GET-NEXT-FEATURE retrieves the next available feature from the set of features remaining to be processed. This procedure has complexity $O(1)$. The next step in the loop is NEIGHBORHOOD-SEARCH. Recall that this procedure retrieves all image features within a neighborhood of the image feature that was obtained by GET-NEXT-FEATURE. This procedure has average complexity $O(\log N)$. Following NEIGHBORHOOD-SEARCH is VERIFY-HYPOTHESES. As described in Sec. 4.3, this procedure decides whether the hypotheses generated by NEIGHBORHOOD-SEARCH are good enough to be considered final hypotheses. This procedure is complexity $O(\log I)$ since it queries the pose tree. Next, the procedure REMOVE-ASSIGNED-FEATURES removes from the set of image features remaining to be processed those features that have been determined by the hypothesis verification stage to belong to a final hypothesis. This is an $O(1)$ step. Finally, a test based on MORE-FEATURES is made. MORE-FEATURES returns TRUE if there are additional image features to be processed by the algorithm and FALSE otherwise. This is also an $O(1)$ step. Thus, the complexity of the loop body is $O(\log I + \log N)$. The loop will be executed, at most, I times (usually considerably fewer than I times). This leads to a combined complexity of $O(I \log I + I \log N)$ for the loop. Combining this with the complexity of the previous four steps yields $O(I \log I + I \log N)$ as the complexity of the entire on-line recognition procedure.

From an information theoretic point of view, the logarithmic dependence of the complexity on the number of model features is a lower bound. This behavior gives us a great deal of latitude to add as many redundant models as necessary of each object in order to achieve good performance without significantly degrading speed of the algorithm. As will be described in the next section, redundant models help to combat widely varying lighting conditions, as well as perspective effects of objects that are really 3D but which the system is treating as 2D.

5. Experimental Results

5.1. Methods

We now present some of the experimental results that we have collected from running our recognition algorithm on a number of images. All images were obtained from a CCD camera at 256×256 resolution. Each pixel was digitized to 8 bits. We then preprocessed all images by applying a Canny edge detector (Canny 1983), and we then applied a simple linking algorithm to trace the contours. After the linking step, we resampled the contours so that both the Cartesian and the θ-s contours were sampled at uniform intervals of arclength. To accomplish this, we used an operation developed in Turney (1986) that simultaneously smoothes the contours, resamples them, and generates both the resampled Cartesian and θ-s contours of the image.

We ran the algorithm on two sets of objects: a set of

Fig. 13. Models of the jigsaw puzzle pieces.

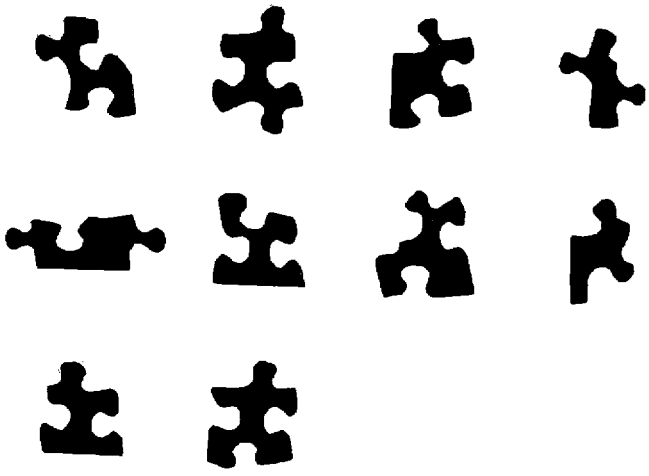
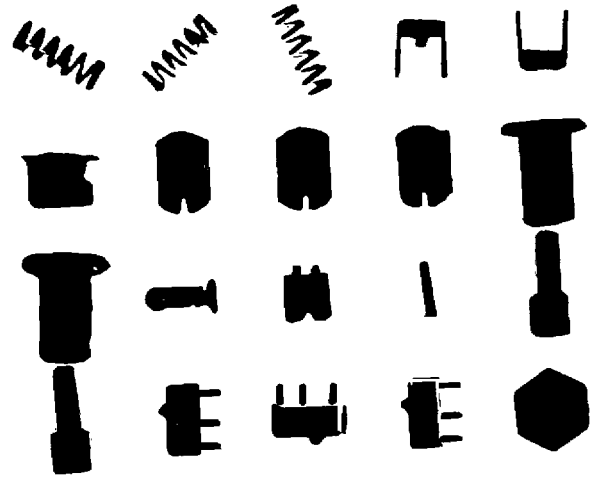


Fig. 14. Models of the switch parts. The microswitch models are shown in the second, third, and fourth positions from the left in the first row.



10 puzzle pieces and a set of eleven switch parts.[5] The puzzle pieces are a set of truly 2D objects that provide a good benchmark. The switch parts are not true 2D parts. However, by treating each view of a distinct stable position as a 2D object, we were able to use our 2D algorithm to recognize the 3D switch parts.

We painted the jigsaw puzzle pieces white to cover the pictures normally appearing on jigsaw puzzles. In all experiments, the parts were scattered randomly in a tray, with some effort made to encourage occlusions. The switch assembly was not painted and was comprised of a number of specular metallic and non-specular plastic parts. The lighting used was a simple fluorescent desk lamp with a movable arm. This was used to reposition the light prior to the acquisition of each image, allowing us to measure robustness to lighting changes.

The algorithm was run on an Apollo series DN570 color workstation, a 5 MIPS machine (roughly).

5.2. Off-line Preprocessing

The off-line processing needs to be performed only once for each distinct set of objects that the system is required to work on. After a training image was preprocessed as previously described, the next step was the assignment of contours to object labels (our training images typically contain several objects). The CPN features of the model contours were then extracted using a simple one-dimensional derivative of a Gaussian edge detector as described in Sec. 4.2. We then applied the K-L expansion to the CPN features to obtain the reduced basis, and the model CPN's were then projected onto the subspace spanned by the reduced basis, also per Sec. 4.2. The projections of the CPN's were stored for use by the on-line recognition procedure.

Figure 13 shows the set of ten puzzle pieces which form our first model set. Similarly, Figure 14 shows the set of views the algorithm will use to form the model set for the switch parts. Note that in Figure 14, some of the models are of the same stable position of a part. It was advantageous to employ these redundant models for a number of reasons. First, the fact that the switch parts are really 3D implies that their aspects change slightly depending on where in the image they are located in the field of view. Secondly, some of the switch parts were metallic and quite reflective. We found that for these parts, lighting changes could alter the shape of some of the critical points enough that the algorithm could not recognize a part (this is especially true of the reflective parts that have few critical points). Adding a model whose training image was taken in different lighting greatly improved the robust-

5. These parts were provided by the U.S. Air Force. They are some of the same parts that Cowan, Chelberg, and Lim (1984) used to test ACRONYM.

Fig. 15. A representative example of running the algorithm on an image of overlapping puzzle pieces. On the left are the contours of an image. On the right are images showing the contours on the left superimposed with images of the final hypotheses filled with patterns so that the various final hypotheses may be distinguished from each other.

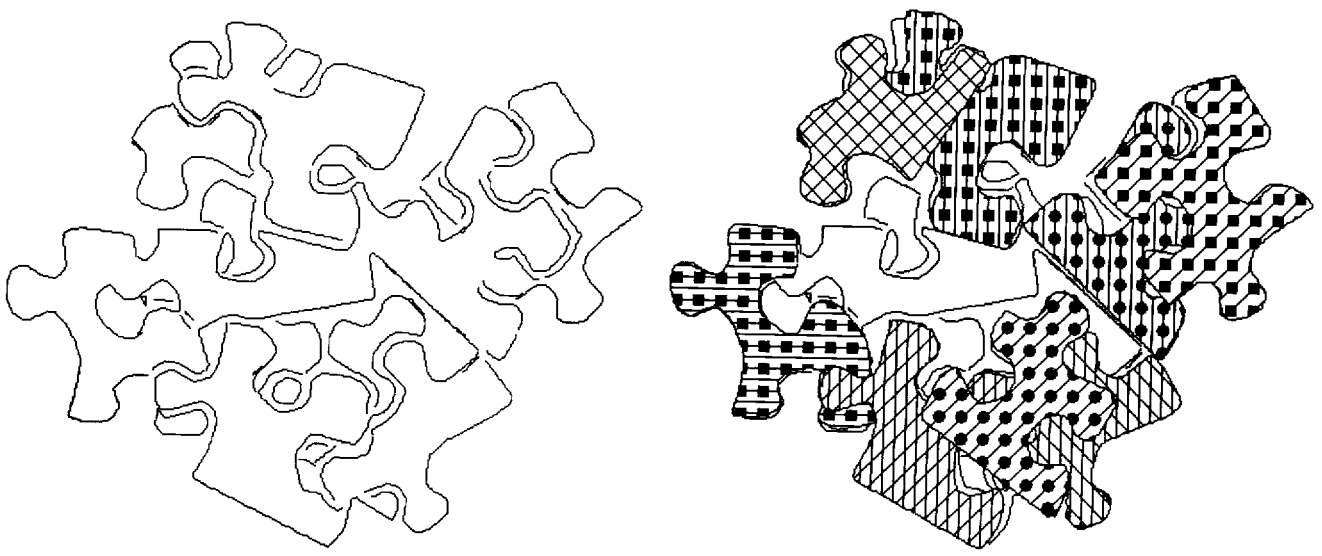


Table 1. False Alarm (F.A.) Statistics for Experiments on the Two Sets of Objects

Object Set	No. Images with 0 F.A.s	No. Images with 1 F.A.	No. Images with 2 F.A.s	Total No. Images
Puzzle parts	6	0	0	6
Switch parts	9	4	1	14

ness of our algorithm with respect to lighting changes for such parts. Finally, and again related to lighting changes, one part (the microswitch—see Fig. 14) had details that were near the limit of our resolution and were extremely sensitive to lighting: images taken in one set of conditions would show the detail, while images taken in other conditions would not show the detail. To solve the problem, we included one model for each case. Because our matching algorithm is only logarithmic in the number of model features, the addition of more views did not significantly affect the runtime of the algorithm. In fact, doubling the number of views would increase the matching by just one step.

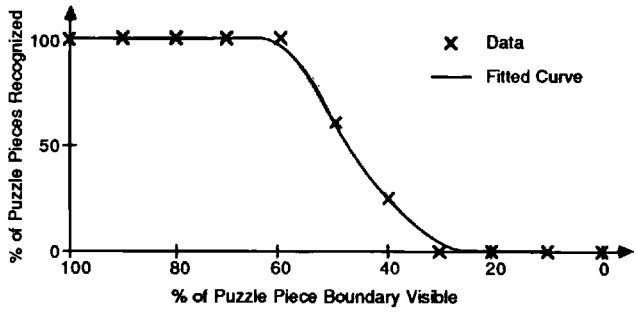
5.3. Recognition Processing and Results

In this section we experimentally assess the accuracy, robustness, and efficiency of our algorithm. We give three means of characterizing the accuracy of our algorithm:

1. Plots of the percentage, p, of the objects that the algorithm recognized correctly versus the percentage of the object's contour that was exposed (Figs. 16 and 18). Note that the percentage of objects missed (i.e., not found by the algorithm when they should have been found) is $100 - p$.
2. The number of false alarms generated in each image (Table 1). This is the number of times that the algorithm predicts an object to be in the image that is not actually there.
3. Images before the recognition procedure is run, and the same images with the final hypotheses superimposed. This yields a reasonable, although qualitative, measure of the positional accuracy of our algorithm.

To characterize the robustness of our algorithm, we

Fig. 16. Percent of objects recognized correctly vs. percent of object boundary visible: jigsaw puzzle pieces. The performance statistics were gathered over a sample of six images.



rely on the fact that we have run a sizable number of experiments under widely differing conditions, namely, two object sets, several lighting setups, and many object placements. In addition, the plot of the percentage of objects recognized correctly versus the percentage of the object's contour visible makes explicit the algorithm's robustness with respect to occlusion. Finally, to characterize the algorithm's efficiency, we give (in addition to the complexity) average runtime of the algorithm from the feature detection step onward for each object set.

After the preprocessing described earlier in this section, the on-line portion of our algorithm proceeds as described in Sec. 4.4.

Puzzle Pieces

Figure 15 shows sample results of running our algorithm on an image of overlapping puzzle pieces, and Figure 16 illustrates the algorithm's robustness with respect to occlusion. It shows a plot of the percentage of puzzle pieces correctly recognized versus the percentage of the boundary of the puzzle pieces exposed. As can be seen from the figure, none of the puzzle pieces is wrongly classified. Also, any puzzle piece with more than 55% of its boundary exposed is recognized correctly. For those pieces with less than 55% of their boundaries exposed, the algorithm sometimes has no hypothesis good enough to consider as a final hypothesis. However, the false alarm numbers in Table 1 show that if the algorithm does have a final hypothesis, it will be correct with near certainty.

The four variables that determine the decision region specified by (6) for the set of puzzle parts are as follows: $\beta = 0$, $T_1 = 0.3$, $T_2 = 0$, and $T_3 = 0$. In other words, for this part set, only Q_{cp} is used to decide whether or not to keep a hypothesis. The reason for this is simply that the puzzle pieces have many critical points, and as discussed in Sec. 4.3, for such objects, Q_{cp} is really the only decision variable needed.

The average time to finish an entire image from the stage of feature detection onward was 2.0 s for test images containing 10 puzzle pieces each. Humans who are familiar with the puzzle pieces generally took at least 20 s to recognize as many puzzle pieces as they could from an image. They could usually recognize more pieces than the algorithm; however, they also averaged more false alarms per image. Humans do better with less boundary visible than the algorithm, presumably because they use other information in addition to boundaries.

Switch Parts

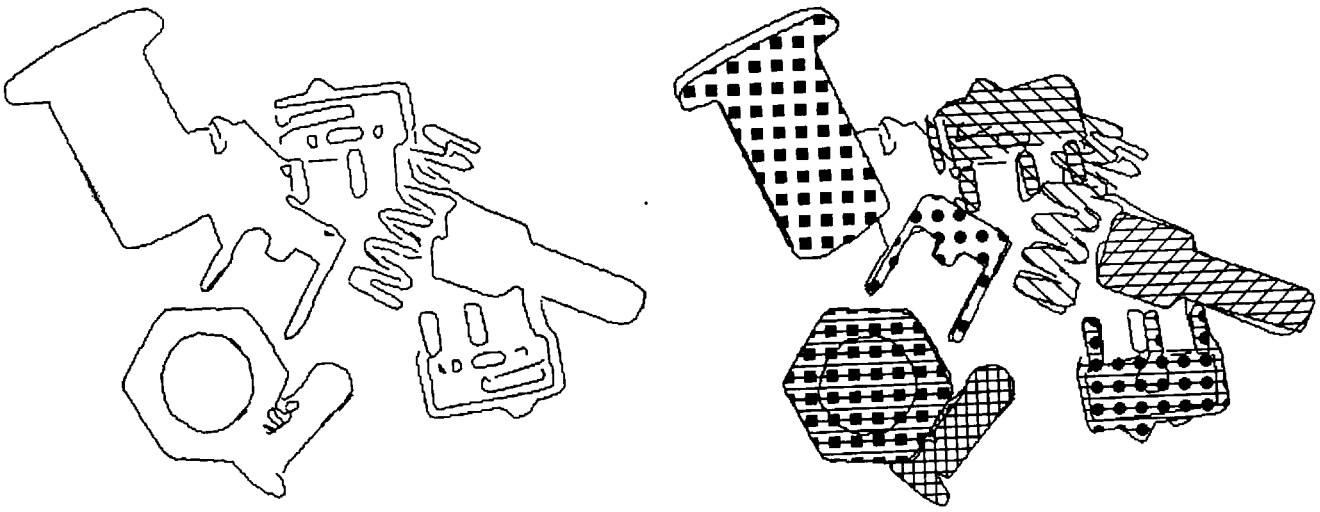
Figure 17 shows a representative example of running our algorithm on an image of overlapping switch parts. Figure 18 gives a plot of the percentage of the switch parts that were recognized correctly versus the percentage of part boundary visible. As can be seen from both figures, the algorithm had more difficulty recognizing the switch parts than it did recognizing the puzzle pieces. Table 1 summarizes the false alarm statistics on the images of the switch parts.

The decision region for the switch parts is given by (6) and the following list of parameter values: $T_1 = 0.3$, $T_2 = 0.3$, $T_3 = 0.4$, $\beta = 0.6$. The runtime of the algorithm on images containing the switch parts averaged 1.5 s, again from the step of feature detection onward.

6. Conclusions

In this paper, we have presented a new procedure for 2D partially visible object recognition. The neighborhoods of critical points were employed as the fundamental features. The heart of the method was the use of a kD tree for fast feature matching. The use of the kD tree was made feasible by applying the Karhunen-Loève expansion to the feature vectors to reduce the

Fig. 17. A sample of the results of running the algorithm on the images of the switch parts. This figure is analogous to Fig. 15.

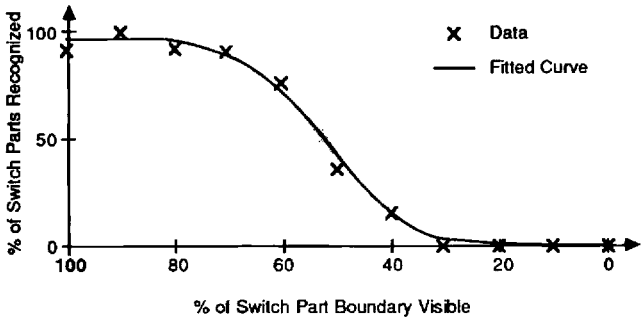


data in them by an order of magnitude. Experiments were conducted on two sets of real objects, jigsaw puzzle pieces and switch parts, to get an idea of the accuracy, robustness, and efficiency of our algorithm. A high degree of accuracy was obtained for objects having more than 50–60% of their boundary exposed. Objects that were more heavily occluded were still recognized much of the time, but success was less certain. Although a direct comparison is inappropriate, it is interesting to compare this with the success one would expect from the analogous problem of recognizing a sentence with half of the letters and spaces missing.

The results of the experiments we have conducted and our experience developing the algorithm have led to an interesting conclusion. Observe first that in a few of the images in Fig. 17, the spring found in the image was shifted one or more cycles from the correct position. This is not surprising, since all of the critical points along the side of the spring are very similar and generate many spurious hypotheses that place the spring shifted from where it should be. Our algorithm occasionally chooses one of the spurious hypotheses, because it may just happen to adjoin a section of boundary from another part, thus making Q_b large enough to pass the false hypothesis over the correct one. In fact, this scenario also leads to most of the false alarms in the experiments. Interestingly, all of these false alarms as well as most of the misplacements of the spring could easily be eliminated if some simple segmentation information was employed in addition to just the shape of the edge contours. In particular, if the background region was known, then these problems could often be eliminated, since in many cases, such false (or poor) hypotheses will have large sections of their contour deep in background with no other contours nearby. This information could be used to weaken those hypotheses, making the correct one more likely to be chosen as a final hypothesis. We believe that employing segmentation information will be necessary to solve the problem in 3D of partially visible object recognition. Our algorithm currently uses no information about the background.

Finally, we note that our recognition system could be extended in a straightforward manner to the recognition of scaled objects if a scale-invariant feature vector that is not sensitive to noise could be found. There are many possibilities employing multiple points. We have found a method that normalizes scale using local curvature information at a single critical point to yield a scale invariant feature vector encoding the local shape of the object (Gottschalk and Mudge 1988). These features show promise for use in recognition of both scaled 2D and 3D objects. Using these observations, we are currently working to extend the method presented here to the domain of 3D partially visible objects. In addition to the edge-based features described here, it is likely that other attributes of objects will be useful in recognition. The kD tree matching technique that we have described is sufficiently general to accommodate such extensions with ease.

Fig. 18. Percent of objects recognized correctly vs. percent of object boundary visible: switch parts. Performance statistics for this figure were gathered from a sample of 14 images.



Acknowledgments

This work was supported in part by AFOSR grant 080012 and ARO grant DAAG29-84-K-0070.

References

Attneave, F. 1954. Some informational aspects of visual perception. *Psycholog. Rev.* 61:183–193.

Ayache, N., and Faugeras, O. D. 1986. HYPER: A new approach for the recognition and positioning of two-dimensional objects. *IEEE Trans. Pattern Anal. Mach. Intell.* 8(1):44–54.

Ballard, D. H. 1981. Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recog.* 13(2):111–122.

Ballard, D. H., and Sabbah, D. 1985. Viewer independent shape recognition. *Proc. IEEE Conf. Pattern Recog. Image Process.*, pp. 67–71.

Bentley, J. L., and Friedman, J. H. 1979. Data structures for range searching. *Comp. Surv.* 11(4):398–409.

Bhanu, B., and Faugeras, O. D. 1984. Shape matching of two-dimensional objects. *IEEE Trans. Pattern Anal. Mach. Intell.* 6(2):137–156.

Biederman, I. 1985, Human image understanding: Recent research and a theory. *Comp. Vision, Graphics, Image Process.* 32:29–73.

Blum, H., and Nagel, R. N. 1978. Shape description using weighted symmetric axis features. *Pattern Recog.* 10:167–180.

Bolles, R. C., and Cain, R. A. 1982. Recognizing and locating partially visible objects: The local-feature-focus method. *Int. J. Robot. Res.* 1(3):57–82.

Bressenham, J. E. 1965. Algorithm for computer control of digital plotters. *IBM Sys. J.* 4(1):25–30.

Canny, J. F. 1983. Finding edges and lines in images. Master's thesis, MIT, Cambridge, Mass.

Chien, C. H., and Aggarwal, J. K. 1987. Shape recognition from single silhouettes. *Proc. IEEE First Int. Conf. Comp. Vision,* pp. 481–490.

Chin, R. T., and Dyer, C. R. 1986. Model-based recognition in robot vision. *Comp. Surv.* 18(1):67–108.

Cowan, C. K., Chelberg, D. M., and Lim, H. S. 1984. ACRONYM model based vision in the intelligent task automation project. *First Conf. AI Appl.*, pp. 176–183.

Dubois, S. R., and Glanz, F. H. 1986. An autoregressive model approach to two-dimensional shape classification. *IEEE Trans. Pattern Anal. Mach. Intell.* 8(1):55–66.

Freeman, H. 1977. Shape description via the use of critical points. *Proc. IEEE Conf. Pattern Recog. Image Process.*, pp. 168–174.

Fu, K. S. 1974. *Syntactic Methods in Pattern Recognition.* New York: Academic Press, 1974.

Goad, C. 1983. Special purpose automatic programming for 3D model-based vision. *Proc. Image Understanding Workshop.* DARPA, pp. 94–103.

Gottschalk, P. G., and Mudge, T. 1988. Efficient encoding of local shape: Features for 3-d object recognition. *Proc. SPIE: Intell. Robots Comp. Vision.* Seventh in a Series, pp. 46–56.

Gottschalk, P. G., Turney, J. L., and Mudge, T. 1987. Two-dimensional partially visible object recognition using efficient multidimensional range queries. *Proc. IEEE Conf. Robot. Automat.*, pp. 1582–1589.

Grimson, W. E. L., and Lozano-Pérez, T. 1987. Localizing overlapping parts by searching the interpretation tree. *IEEE Trans. Pattern Anal. Mach. Intell.* 9(4):469–482.

Knoll, T. F., and Jain, R. C. 1986. Recognizing partially visible objects using feature indexed hypotheses. *IEEE J. Robot. Automat.* 2(1):3–13.

Koch, M. W., and Kashyap, R. L. 1985. A vision system to identify occluded industrial parts. *Proc. IEEE Conf. Pattern Recog. Image Process.*, pp. 55–60.

Lamdan, Y., Schwartz, J. T., and Wolfson, H. J. 1988. On recognition of 3-d objects from 2-d images. *Proc. IEEE Conf. Robot. Automat.*, pp. 1407–1413.

Lowe, D. G. 1987. The viewpoint consistency constraint. *Int. J. Comp. Vision* 1(1):58–72.

Mattill, J. 1976. The bin of parts problem and the ice-box puzzle. *Technol. Rev.* 78(7):18–19.

McKee, J. W., and Aggarwal, J. K. 1977. Computer recognition of partial views of curved objects. *IEEE Trans. Comp.* 26(8):790–800.

Mudge, T. N., Turney, J. L., and Volz, R. A. 1987. Automatic generation of salient features for the recognition of partially occluded parts. *Robotica,* 5:117–127.

Pavlidis, T. 1977. *Structural Pattern Recognition.* New York: Springer.

Perkins, W. A. 1978. A model-based vision system for industrial parts. *IEEE Trans. Comp.* 27(2):126–143.

Rosenfeld, A., and Kak, A. C. 1976 *Digital Picture Processing.* New York: Academic Press, pp. 109–123.

Segen, J. 1983 (November). Locating randomly oriented shapes from partial views. *Proc. Soc. Photo-optical Instrum. Eng. Cambridge Symp. Robot Vision and Sensory Controls,* pp. 676–684.

Schwartz, J. T., and Sharir, M. 1986. Two-dimensional, model based boundary matching using footprints. *Int. J. Robot. Res.* 5(4):38–55.

Tropf, H. 1980. Analysis-by-synthesis search for segmentation applied to workpiece recognition. *Fifth IEEE Int. Conf. Pattern Recog.,* pp. 241–244.

Tsui, H. T., and Chan, M. H. 1987. Recognition of partially occluded two-dimensional objects. *Comp. Digit. Tech., IEEE Proc.* 134(1):9–16.

Turney, J. L. 1986. Recognition of partially occluded parts. Ph.D. dissertation, Department of Electrical Engineering and Computer Science, University of Michigan.

Turney, J. L., Mudge, T., and Volz, R. A. 1985. Recognizing partially occluded parts. *IEEE Trans. Pattern Anal. Mach. Intell.* 7(4):410–421.

Turney, J. L., Mudge, T., and Volz, R. A. 1986. Solving the bin of parts problem. *Vision 86 Conf. Proc.,* pp. 4-21–4-38.

Yam, K. R., Martin, W. N., and Aggarwal, J. K. 1980. Analysis of scenes containing several occluding curvilinear objects. TR-135. Austin, Texas: University of Texas at Austin.