Two-Dimensional Partially Visible Object Recognition Using Efficient Multidimensional Range Queries[1]

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

Robotics Research Laboratory, EECS Department, University of Michigan, Ann Arbor, MI, 48109

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 less efficient than is possible. This is due primarily to insufficient attention being paid to the issues of reducing the data in features and feature matching. In this paper we discuss an algorithm that addresses both of these problems. Our algorithm uses the local shape of contour segments near critical points, represented in slope angle-arclength space (θ-s space), as the fundamental feature vectors. These fundamental feature vectors are further processed by projecting them onto a subspace of θ-s space that is obtained by applying the Karhunen-Loève expansion to all critical points in the model set to obtain the final feature vectors. This allows the data needed to store the features to be reduced, while retaining nearly all their recognitive information. The resultant set of feature vectors from the image are matched to the model set using multidimensional range queries to a database of model feature vectors. The database is implemented using an efficient data-structure called a k-d tree. The entire recognition procedure for one image has complexity $O(I \log I + I \log N)$, where I is the number of features in the image, and N is the number of model features. Experimental results showing our algorithm's performance on a number of test images are presented.

1 Introduction

A problem which has received considerable attention in the computer vision literature is that of recognizing two-dimensional (2-d) 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 (3-d) partially visible objects in an image. The problem of recognition of partially visible objects is sometimes called the *bin of parts problem* after the way that parts are commonly presented for batch assembly in industry: 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 [9]. In this paper, we present a solution to the bin of parts problem where the objects are 2-d or have a small number of aspects (and hence are essentially 2-d).

There are three very general goals which should be common to all object recognition systems: accuracy, robustness, and efficiency. The 2-d partially visible object recognition algorithm presented here is highly accurate, robust, and efficient. In addition to being a novel approach to the problem of 2-d partially visible object recognition, we will argue that our algorithm achieves the goals of accuracy, robustness, and efficiency to a greater extent than many previous algorithms. For a thorough review of previous work relating to 2-d object recognition, the reader is referred to [15] and [8]. While no work we know of combines all of the elements of our present work, a number of previous works have one or more similar aspects. Freeman [7], like us, uses critical points as his fundamental features (critical points are discussed in detail later). He augments this set with other types of geometrical features, which we do not use, such as end points, intersections, and points of tangency. Perkins [11], Yam *et al.* [16], Mckee *et al.* [10], and Turney [15] have all employed the slope angle–arclength representation of edges (discussed later). The features and the matching strategies used in these works differs considerably from ours, however. Bolles and Cain [3] have used highly informative features to advantage, as has Turney [15]. A portion of our hypothesis verification algorithm is similar to the hypothesis verification schemes in [3] and [8].

2 Two-Dimensional Object Recognition

Conceptually, the simplest strategy for recognizing 2-d 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 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 which has the desirable

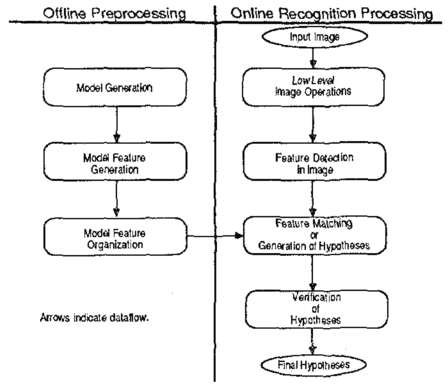
Offline Preprocessing
Online Recognition Processing
Input Image
Model Generation
Low Level Image Operations
Model Feature Generation
Feature Detection in Image
Model Feature Organization
Feature Matching or Generation of Hypotheses
Verification of Hypotheses
Arrows indicate dataflow.
Final Hypotheses

Figure 1: An Abstract View of the Recognition Process

property of greatly reducing the data needed to represent the image adequately. At the same time, extracting a feature should sacrifice as little of the recognitive information in the image as possible. It is desired (and often true) that the time taken to extract the features from the image is more than made up by the reduced time needed to perform recognition. 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. Any global features would be too prone to distortion if an object in the scene was occluded by another.

Figure 1 shows the general strategy which many recognition procedures use. Examining the online half of Figure 1, the input image may be 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 features. The features may be represented by a 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

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

vector[2]. These initial hypotheses must then be verified or rejected. The hypotheses that pass the final verification phase are those which are most likely to hypothesize the correct object appearing at the correct pose in the image.

The offline preprocessing branch of Figure 1 starts with a model generation phase. Models are typically generated by processing a set of training images or by a CAD system [13]. 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 step in offline processing, the model features may be organized into a data structure which can then be accessed during the feature matching phase of the recognition procedure.

A very important aspect of the recognition process is the relationship between the offline step of model feature organization and the online step of feature matching (see Figure 1). A very powerful means of speeding up the feature matching stage is available through the organization of the data structure that the model feature organization stage creates; this data structure should be organized for the fastest possible retrieval of the matching model features. Some algorithms have taken a step in this direction. For example, Turney [14] sorts the set of model features by their saliency (saliency formalizes the notion of informativeness of 2-d features). However, the matching process is still a linear search. Knoll and Jain [8] 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 such 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 which match a feature vector from the image as an an abstract data type: given a K-dimensional vector, return a list of all K-dimensional vectors from a set of such vectors that lie within a K-dimensional neighborhood (as defined by some distance metric) of the original vector. 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 [2]. This is in contrast with the $O(N)$ time of a linear search. In Section 3, we discuss how one of these data structures, a k-d tree, can be used to do very fast feature matching.

3 A New Two-Dimensional Object Recognition Algorithm

3.1 Selection and Computation of Feature Vectors

Our algorithm recognizes objects entirely by the shape of contours. A contour consists of edge points which have been linked together into a single edge segment and stored in a data structure, typically a linked list. We chose to represent the contours in slope angle-arclength space (θ-s space hereafter) as opposed to a Cartesian representation. In the θ-s representation of a contour, the ordinate is the slope angle at the point of interest on the contour (θ), while the abscissa is the arclength (s) measured from some point of reference on the contour to the point of interest.

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

[2]A hypothesis, as we shall use the term, consists of two parts: first, a conjecture as to the identity of an object in the image and second, a conjecture as to the pose of the object in the image. In more formal terms, (and the way in which we shall use the word) a hypothesis is an ordered pair $(\mathcal{M}, \mathcal{T})$ where $\mathcal{M}$ is a model which is hypothesized to appear in the image, and $\mathcal{T}$ is a transformation which is applied to the model in order to place it at the hypothesized pose. In our case the transformation $\mathcal{T}$ is always chosen so that the pose of the model feature, F_m, which matches an image feature, F_i, are as close as possible, according to some metric. We shall, in order to facilitate the discussion, often speak of a hypothesis to be the model $\mathcal{M}$ after applying $\mathcal{T}$ to it.

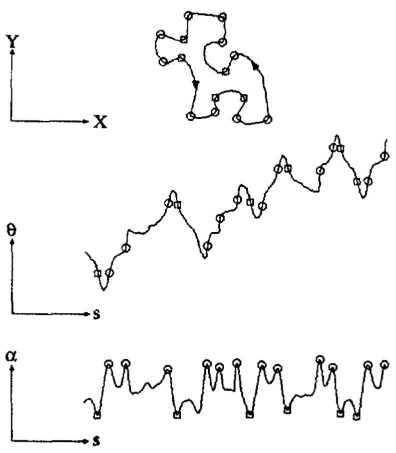

Figure 2: Shown above is the puzzle piece contour represented in both Cartesian space (top) and θ-s space (bottom). Marked in both curves are the location of critical points. Positive extrema of curvature are marked with circles, and negative extrema of curvature are marked with squares. As mentioned in the text, the critical points are the edge points marked by a one-dimensional derivative of a Gaussian edge detector applied to the θ-s representation of the contour.

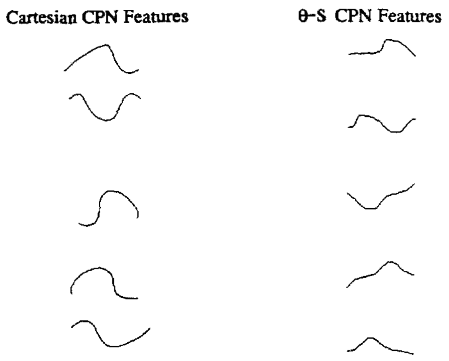

Figure 3: Above on the left are several typical puzzle piece CPN features in their Cartesian representation, and above on the right are the same features represented in θ-s space. Note the similarity of the θ-s *representations.*

is added to θ so that the reference point on the contour is some standard value, such as zero. If this is done, then the θ-s representations of rotated versions of some Cartesian contour are the same. Other advantages of the θ-s representation, such as singlevaluedness 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 our algorithm are discussed. See [13] for a more detailed exposition of the θ-s representation and its properties.

The choice 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 use 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. A critical point is defined to be a point on the contour of an object where there is an extrema of curvature. Attneave [1] and Turney [13] have argued that contour segments which contain critical points have high information content relative to those contour segments comprising an object which do not contain any critical points. An additional advantage that critical points possess is that they are easily and quickly extracted by the application of a one-dimensional edge operator to the θ-s representation of the contours (by definition, a critical point is a point of rapidly changing slope on the θ-s curve representing a contour). It is for the reasons given above that we have chosen to use critical point neighborhoods as the fundamental geometric features in this work.

Figure 2 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 3 reveals that many CPN's look similar. This indicates that samples in a contour of a CPN feature are highly correlated with each other, i.e., it is possible to predict with high accuracy what the value of a sample will be given the values of some other samples on the CPN's contour. The Karhunen-Loève (K-L) expansion, a standard data compression technique in signal processing, takes advantage of highly correlated data. Rosenfeld and Kak [12] 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.

The form of the K-L expansion we shall use applies to real random vectors. Therefore, in order to apply it to CPN features, we first must view them as vectors. The θ-s representation of CPN features is a single valued function of one variable. In practice the representation is discretized and contains a finite number of samples. Each sample 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 $\underline{X}$ of dimension N. Let $\mathbf{R} = E[\underline{X}\,\underline{X}^t]$ be the auto-correlation matrix of $\underline{X}$. Since $\mathbf{R}$ is non-negative definite, there exists a set of orthonormal eigenvectors and associated eigenvalues of $\mathbf{R}$, $\underline{\phi}_k$ and $\lambda_k \geq 0$ respectively, where $k = 1, \ldots, N$. Define the random variables $Y_k = \underline{\phi}_k^t\,\underline{X}$. Without loss of generality, we may assume that the eigenvectors $\underline{\phi}_k$ are ordered so that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_N$. Examining the cross correlations of the Y_k, we find

$$E[Y_kY_l] = E[(\underline{\phi}_k^t\,\underline{X})(\underline{\phi}_l^t\,\underline{X})] = E[(\underline{\phi}_k^t\,\underline{X})(\underline{\phi}_l^t\,\underline{X})^t] = \underline{\phi}_k^t E[\underline{X}\,\underline{X}^t]\underline{\phi}_l = \underline{\phi}_k^t\mathbf{R}\underline{\phi}_l = \lambda_l\delta_{kl} \quad (1)$$

where δ_{kl} is the Kronecker delta function. This implies that for $k \neq l$, Y_k and Y_l are orthogonal random variables. If $\underline{X}$ is zero mean, then

$$E[Y_k] = E[\underline{\phi}_k^t\,\underline{X}] = \underline{\phi}_k^t E[\underline{X}] = 0. \quad (2)$$

Therefore, when $\underline{X}$ is zero mean, we have from (1) and (2) that

$$E[Y_kY_l] = 0 = E[Y_k]E[Y_l],\ k \neq l. \quad (3)$$

This implies that when $k \neq l$, Y_k and Y_l are uncorrelated in addition to being orthogonal. When $k = l$, we get that

$$E[Y_k^2] = \mathrm{Var}(Y_k) = \lambda_k. \quad (4)$$

With the $\underline{\phi}_k$ as the K-L basis, and the random variables Y_k as the coefficients, we arrive at the Karhunen-Loève expansion of $\underline{X}$:

$$\underline{X} = \sum_{k=1}^{N} Y_k\underline{\phi}_k. \quad (5)$$

The mathematics in the preceding paragraph do not clearly illuminate the reasons that the K-L expansion works well as a data compression technique. Geometrically, the K-L expansion chooses a special basis in the N-dimensional vector space in which $\underline{X}$ is defined. This basis has the following property: the basis vector $\underline{\phi}_1$ defines the direction in which $\underline{X}$ has the greatest variance (i.e. Y_1, the projection of $\underline{X}$ in the $\underline{\phi}_1$ direction, has the maximum variance); the second basis vector $\underline{\phi}_2$ defines the direction in the subspace perpendicular to $\underline{\phi}_1$ in which $\underline{X}$ has the greatest variance, and so on until all dimensions are defined. Therefore, the K-L expansion chooses a basis which, when $\underline{X}$ is represented in terms of it, will concentrate the total variance of $\underline{X}$ into its lower numbered components. A subspace whose basis vectors $\underline{\phi}_k$ are associated with those Y_k having small variance may be ignored with negligible effect upon the probabilistic properties of $\underline{X}$. 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.

Before applying the K-L expansion to compress the data needed to represent the CPN features, it is necessary to make the data zero mean, since this decorrelates the Y_k. Let $S = \{\underline{F}_i',\ i = 1, \ldots, V\}$ 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 $T = \{\underline{F}_i = \underline{F}_i' - \underline{M},\ i = 1, \ldots, V\}$, where $\underline{M}$ is the sample mean of S. $\mathbf{R}$ is estimated from T by the formula

$$\mathbf{R} = 1/V \sum_{k=1}^{V} \underline{F}_k^t\,\underline{F}_k. \quad (6)$$

Using the estimate[3] of $\mathbf{R}$, the K-L expansion formulae can then be applied to obtain estimates of the basis vectors $\underline{\phi}_k$ as well as estimates of the variances λ_k of the uncorrelated random variables Y_k. The data reduction is effected by retaining only those basis vectors $\underline{\phi}_k$ associated with the Y_k having the largest variances. Define the total variance of $\underline{X}$, σ_x, as $E[\underline{X}^t\underline{X}]$. Note also that $\sigma_x = \sum_{k=1}^{N}\lambda_k$, since the $\underline{\phi}_k$ are an orthonormal set. The number of $\underline{\phi}_k$'s retained, L, is determined by the fraction of σ_x we wish to retain[4]. The reduced feature vector, $\underline{R}$, of any CPN feature $\underline{F}$ can then be computed by the formula $\underline{R} = \mathbf{P}(\underline{F} - \underline{M})$, where $\mathbf{P}$ is the $L \times N$ matrix whose rows are the L retained $\underline{\phi}_k$ ordered such that the $\underline{\phi}_k$ with the largest associated variance appears at the top, the $\underline{\phi}_k$ with the second largest variance appears second from the top, and so on to the bottom where the last retained $\underline{\phi}_k$ (associated with the smallest variance) appears.

Figure 4 shows the K-L basis vectors, $\underline{\phi}_k$, which have been computed using the entire set of CPN features from the training set, the associated variances of the Y_k, and the reduced feature vectors of the sample of CPN features. In our case, 95% of the total variance was retained; only

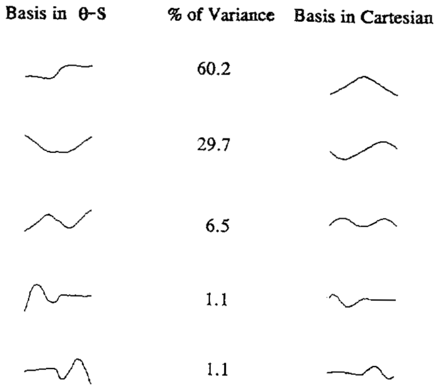

Figure 4: Above on the left are the basis vectors which result from applying the K-L expansion to all of the CPN features in the model set (in this case, a set of ten puzzle pieces) are represented in θ-s space. In the middle of the figure is the percentage of the total variance associated with each basis vector. At the far right are the Cartesian representations of the basis vectors.

5 dimensions out of an initial total of 45 were necessary to achieve the 95% variance figure. This is a reduction in data by nearly an order of magnitude. Figure 5 shows the projections of some typical CPN features onto the reduced basis obtained from applying the K-L expansion to the CPN's of a set of puzzle piece contours.

[3] We shall use the same symbols for both quantities and their estimates.

[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 but 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. 95%) still yields a large data reduction.

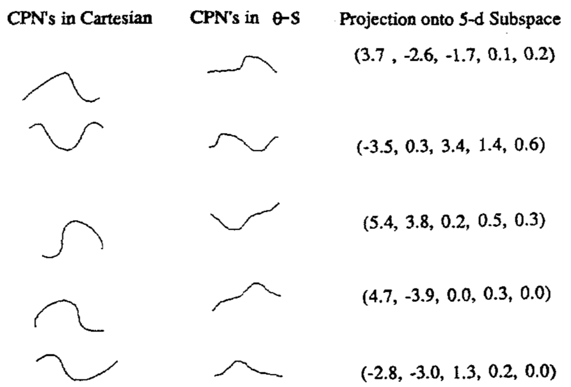

Figure 5: On the left are shown some typical puzzle piece CPN features in Cartesian space. In the middle are their θ-s representations. On the right are their projections onto the reduced basis made up of the five vectors shown on the left in Figure 4.

3.2 Feature matching

The step following the computation of the reduced feature vectors from the image is that of matching those vectors to the reduced feature vectors computed from the training images. Since, even for small object set such as those used in our experiments, there are several hundred reduced model feature vectors to be compared with each reduced image feature vector, it is crucial that the feature matching method be as efficient as possible.

It was mentioned briefly in Section 1 that the problem of matching can be stated as a problem in near neighbor retrieval from a set: given a set of K-dimensional vectors, retrieve all those vectors in the set which fall within a K-dimensional neighborhood of another vector. We shall henceforth call this operation a neighborhood search[5]. The definition of neighborhood includes the choice of a distance metric. One general class of metrics is defined by

$$D_n(\underline{a},\underline{b}) = \left[\sum_{i=1}^{K} |a_i - b_i|^n\right]^{1/n} \tag{7}$$

where K is the dimensionality of the vectors $\underline{a}$ and $\underline{b}$, a_i and b_i are the components of $\underline{a}$ and $\underline{b}$ respectively, and n is the order of the metric. A special case of (7), the D_∞ metric, or Chebeychev metric, is obtained as $n \rightarrow \infty$ and can also be written as

$$D_\infty(\underline{a},\underline{b}) = \max_i |a_i - b_i| \tag{8}$$

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

There are a number of data structures known that may be used to perform the range searching operation [2]. The k-d tree was chosen for our purposes, since it has the best average case query time, preprocessing time, and space requirements in addition to being the easiest to code.

A k-d tree is a binary tree with two pieces of additional information stored at each node, a key identifier and a discrimination value. 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 k-d tree is stored in its leaves. The leaves are lists of less than some predetermined length which contain data satisfying the constraints of all of the ancestor nodes.

In our application, the k-d 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 k-d tree during its construction unless the data is highly degenerate[6]. 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 (organizing the tree in this way makes queries more efficient [2]). 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 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 procedure terminates 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 is less than J records left in the set, a leaf, which is a linked list of less than J elements, is formed.

Searching a k-d 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 procedure terminates when a leaf is encountered. In this case, the list making up the leaf is scanned and any records whose keys do not fall in all of the range intervals are excluded. A linked list of the remaining records is returned.

We have shown that the k-d 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 (7) with no increase in complexity for either queries, space, or preprocessing. It was noted previously that for a given δ, $N(\delta,\underline{s},\infty)$ contains $N(\delta,\underline{s},n)$. Thus, all that must be done to implement a neighborhood search for finite n is to scan the result of a $N(\delta,\underline{s},\infty)$ search implemented by a K-dimensional range search using a k-d tree and exclude those vectors that do not fall into $N(\delta,\underline{s},n)$. The complexity of a query remains the same as the $N(\delta,\underline{s},\infty)$ case, because the search via the k-d tree is an $O(\log N + F)$ operation, and the scanning complexity is $O(F)$, the query complexity remains $O(\log N + F)$ (recalling that N is the size of the set to be searched, and F is the size of the retrieved set) even with the addition of the exclusion step.

Feature matching in our algorithm is a neighborhood query of the set of reduced CPN feature vectors from the model using the Euclidean neighborhood $N(\delta,\underline{R}_I,2)$, where $\underline{R}_I$ is the reduced image feature we are matching against. The value of δ is adjusted so it is just large enough so that, for most features, the correct match will be generated amongst the other matches if the CPN feature in the image is unobscured. We have found, in the object sets we have tested, that there are usually three or four model features in the neighborhood of a given image feature. Each of these matches generates a hypothesis that will need to be tested by

[5]Note that there are two distinct ways in which we have used the word neighborhood. The most recent usage in the phrase "neighborhood search" is a common usage in mathematical analysis. It refers to all vectors in a vector space which are within some distance of another vector. The metric, or distance measure, is ours to define as long as it obeys some basic rules. The other, quite different, usage occurs in the phrase "critical point neighborhood". This usage refers to an interval of arclength in the θ-s representation of a contour which has a critical point at its center. The context should make the usage clear.

[6]The degeneracy occurs when two or more of the keys of two distinct records have the same value. Since the keys, in our case, are continuous variables, the probability of this occurring is infinitesimal.

the method described in Section 3.3. Recall that a hypothesis can be considered as an ordered pair of a model (which we hypothesize gave rise to the image feature) and a pose transformation (which we hypothesize will transform the model to the correct image location so as to give rise to the image feature). When a model feature and an image feature match, a hypothesis is generated as follows: the model which contains the matching feature is taken to be the hypothesized model, and the pose transformation necessary to bring the matched model feature into alignment with the image feature is designated to be the transformation part of the hypothesis. The details of this process are discussed in Section 3.3.

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 k-d trees. In the following section, we discuss our approach to hypothesis verification, and, in particular, we show how k-d trees may also be used to some advantage there.

3.3 Hypothesis Verification

We now turn to the problem of hypothesis verification. Typically, verification of a hypothesis 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 which rates the strength of the hypothesis. The score depends in some way upon how closely what actually appears in the image matches what is expected to appear according to the hypothesis. All the available hypotheses are checked, and the strongest (according to some criterion) are kept. These remaining hypotheses are the algorithm's best guess as to which objects appear in the image, and at what pose they appear.

In terms of the discussion in Section 3.2, hypotheses are those reduced CPN feature vectors within a neighborhood of some image feature vector, together with some additional information: a pointer to the model of the object which contains the feature, and the position of the feature within the model. The model of the object is the set of all of its 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 which 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. Note that the pose transformation of the contours is nothing more than the addition of an offset to the θ values in the θ-s representation of the contours; the offset is chosen to bring the matched CPN from the model of the object into alignment with the image CPN.

Our verification scheme bases its decision about whether to pass or fail a hypothesis upon two statistics: a fraction of critical points matched Q_{cp}, and a fraction of boundary matched Q_b. We chose these two statistics since past experience showed them to be effective decision variables for a wide variety of objects. 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 objects, such as nails, have relatively few critical points. In these cases employing Q_b in conjuction with Q_{cp} is appropriate since these objects often have only one or two critical points visible; all the rest may be occluded by other objects. In addition, Q_{cp} becomes very sensitive to accidental matches when there are few critical points on the object. Any contour segments on such objects are roughly equally informative, hence the presence of any portions of the hypothesized contour in the image can be regarded as evidence that the hypothesized object was indeed present in the scene.

We now detail the computation of the statistics. 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[7]. 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 contour at the critical point (recall that we consider the pose transformation to have already taken place). Q_{cp} is given by

$$Q_{cp} = I_{cp}/M_{cp}, \tag{9}$$

where I_{cp} is the number of model critical points in the hypothesis which 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 k-d tree is employed. For reasons that will shortly become apparent, we shall call this k-d 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 (i.e. the ambiguity as to the value of the integer l in the equation $\theta_k = \arctan(m) + 2\pi l$, where m is the slope of the tangent at the critical point). 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 4-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 .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, and Q_{cp} may then be computed.

We have discussed the computation of the 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. 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). Following the step of 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 [4] such that the line probed is perpendicular to the hypothesis contour, and the pixels in the line are checked up to 2 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 pass or fail a hypothesis. As would be expected, the optimal decision regions depend upon the object set as well as the degree of occlusion allowed. While we do not find optimal decision regions, we nevertheless desired to be general enough to get good performance for a wide variety of object sets and degrees of occlusion. In particular, a hypothesis is passed if the ordered pair (Q_{cp}, Q_b) falls into the following region and is failed otherwise:

$$\{(x,y) : x > T_1,\ y > T_2, \text{ and } \beta x + (1-\beta)y > T_3\}. \tag{10}$$

[7]The matching we are discussing presently in reference to hypothesis verification is distinct from the feature matching process discussed in Section 3.2

The three thresholds and β are chosen to give the best performance with the given object set.

3.4 Summary of the Algorithm

The previous three sections have dealt in detail with the heart of our algorithm. In this section, we shall summarize the algorithm and analyse its complexity.

Let N be the number 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 offline computation first, ignoring the low level operations of edge detection and edge linking. The offline 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 CPN's, and building of the k-d tree. These steps have complexity $O(N) = O(N/b)$, $O(N)$, $O(N)$, $O(1)$, $O(N)$, and $O(N \log N)$ respectively. Thus the entire procedure has complexity $O(N \log N)$.

The online portion of the algorithm is comprised 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 online portion of the algorithm in Pascal-like pseudocode to make it as clear as possible.

```
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 preceding discussion. They have all been discussed previously. 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 complexity $O(\log N)$. Following NEIGHBORHOOD-SEARCH is VERIFY-HYPOTHESES. As described in Section 3.3, 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 otherwize. 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 a cumulative 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 online recognition procedure.

4 Experimental Results

We now present some of the experimental results which 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 from a CCD camera. Each pixel was digitized to 8 bits of gray scale. We then preprocessed all images by applying a Canny edge detector [6], 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 [15] which simultaneously smoothes the contours, resamples them, and generates both the resampled Cartesian and θ-s contours of the image.

4.1 Offline Processing

The offline 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 described above, 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 Section 3.1. 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 Section 3.1. The projections of the CPN's were stored for use by the online recognition procedure.

We ran the algorithm on two sets of objects: a set of ten puzzle pieces and a set of eleven switch parts[8]. The puzzle pieces are a set of truly 2-d objects which provide a good benchmark. The switch parts are not 2-d parts, however. We allow only the stable positions of the switch parts to occur in an image. By treating each view of a distinct stable position as a 2-d object, we were able to use our 2-d algorithm to recognize the 3-d switch parts.

All of the experiments were conducted using an Apollo series 660 color workstation. Run times on this machine are roughly equivalent to run times on a VAX 11/750.

4.2 Online 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. A plot of the percentage of the objects which the algorithm recognized correctly (as determined by a human observer) versus the percentage of the object's contour which is exposed.

2. The number of false alarms generated in each image. This is the number of times that the algorithm predicts an object to be in the image which is not actually there (again determined by a human observer).

3. Figures of some of images before the recognition procedure is run, and figures of the first image with the final hypotheses of the algorithm superimposed in differing gray tones. This yields a reasonable, although qualitative, measure of the positional accuracy of our algorithm.

To characterize the robustness of our algorithm, we 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, ie. we do not include the low level operations of edge detection and edge linking in the runtime figures.

After the preprocessing described earlier in this section, the online portion of our algorithm proceeds as described in Section 3.4.

4.2.1 Puzzle Pieces

Figure 6 shows the result of running our algorithm on an image of overlapping puzzle pieces. The left hand side of each figure is a picture

[8]These switch parts were provided by the Air Force. They are some of the same parts that Cowan *et al.* [5] used to test ACRONYM.

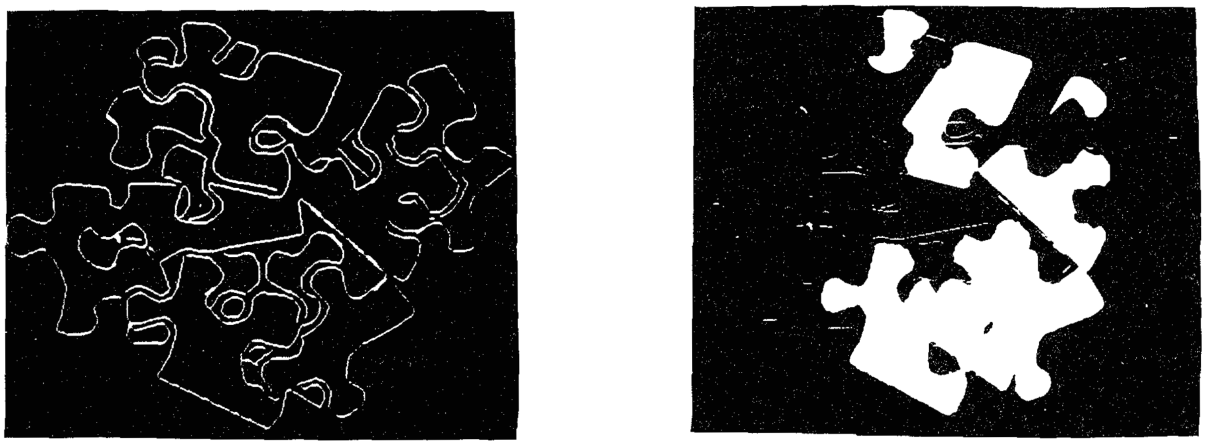

Figure 6: Above are the results of running the algorithm on an image of puzzle pieces. On the left are the contours of an image (the contours are white). On the right are images showing the image on the left superimposed with filled gray tone images of the final hypotheses.

of the puzzle piece contours that were extracted from the image. On the right, filled in differing shades of gray, are the algorithm's final hypotheses as to the identity and location of the various puzzle pieces in the image. We ran the algorithm on a total of six different images containing puzzle pieces. The example shown in Figure 6 was chosen as typical. The algorithm generated no false alarms for any of the images containing puzzle pieces. Figure 7 summarizes the algorithm's performance

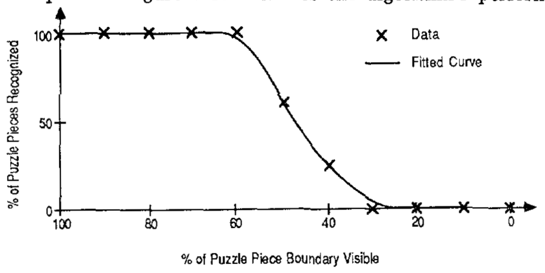

Figure 7: Percent of Objects Recognized Correctly vs. Percent of Object Boundary Visible: Jigsaw Puzzle Pieces.

and, in addition, makes explicit its robustness with respect to occlusion. Shown there is a plot of the percentage of puzzle pieces correctly recognized versus the percentage of the boundary of the puzzle pieces exposed. Figure 7 shows that any puzzle piece with more than 50% of its boundary exposed will, with high likelihood, be recognized correctly. For those pieces with less than 50% of their boundaries exposed, the algorithm sometimes has no hypothesis good enough to consider as a final hypothesis. However, the fact that the number of false alarms is so small shows that if the algorithm does have a final hypothesis, it will be correct with near certainty.

The four variables which determine the decision region specified by (10) for the set of puzzle parts are as follows: $\beta = 0$, $T_1 = .3$, $T_2 = 0$, 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 Section 3.3, 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 5.3 seconds for test images containing ten puzzle pieces each. Humans who are familiar with the puzzle pieces generally took at least 20 seconds to recognize all the puzzle pieces they could from an image. However, they could usually recognize more pieces (i.e. humans do better with less boundery visible than the algorithm). They also averaged more false alarms per image than the algorithm did.

4.2.2 Switch Parts

Figure 8 shows the results of running our algorithm on two images of overlapping switch parts. Experiments were conducted on a total of fourteen images conating overlapping switch parts. As in the case of the puzzle pieces, the images were chosen as typical examples of the algorithm's performance. When processing four of the images, a single false alarm was generated. The other ten images had no false alarms. Figure 9 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 Figure 8 and Figure 9, the algorithm had more difficulty recognizing the switch parts than it did recognizing the puzzle pieces.

The decision region for the switch parts is given by (10) and the

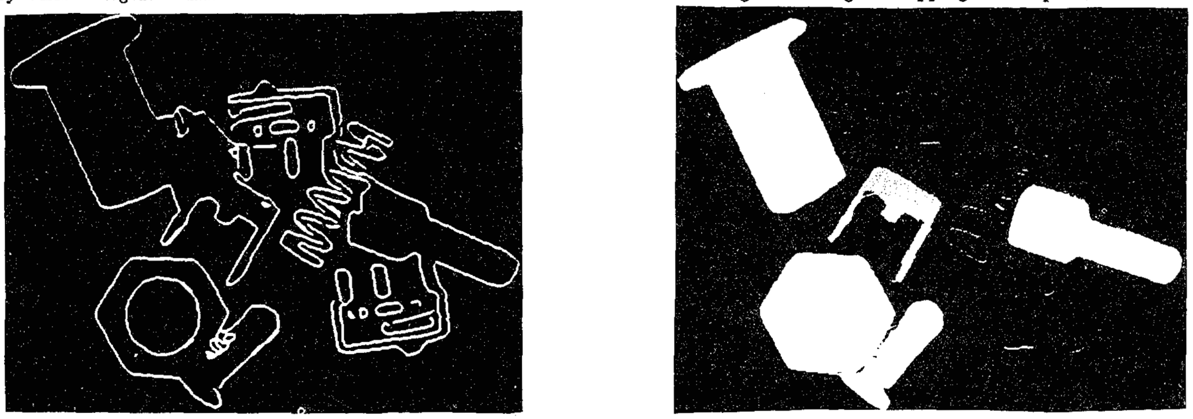

Figure 8: Above are the results of running the algorithm on an image of the switch parts. This figure is exactly analogous to Figure 6.

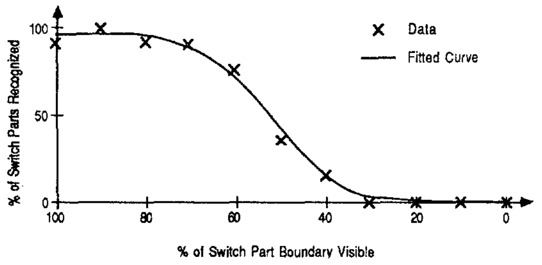

Figure 9: Percent of Objects Recognized Correctly vs. Percent of Object Boundary Visible: Switch Parts

following list of parameter values: $T_1 = .3$, $T_2 = .3$, $T_3 = .4$, $\beta = .6$.

The runtime of the algorithm on images containing the switch parts averaged 4.8 seconds, again from the step of feature detection onward.

5 Conclusions

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

The results of the experiments we have conducted and our experience developing the algorithm has led to an interesting conclusion. In a few of the images of the switch parts, the spring was found in the image 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 false hypotheses which place the spring shifted from where it should be. Our algorithm occasionally chooses one of the false 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. If this information were available, it 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 3-d of partially visible object recognition. Our algorithm currently uses no information as to what is background and what is not. Using these observations, we are currently working to extend the method presented here to the domain of 3-d partially visible objects.

References

[1] F. Attneave, "Some Informational Aspects of Visual Perception," *Psychological Review*, Vol. 61, pp. 183-193, 1954.

[2] J. L. Bentley and J. H. Friedman, "Data Structures for Range Searching," *Computing Surveys*, Vol. 11, No. 4, December 1979.

[3] R. C. Bolles and R. A. Cain, "Recognizing and Locating Partially Visible Objects: The Local-Feature-Focus Method," in *Robot Vision*, A. Pugh, Ed., 1984.

[4] J. E. Bressenham, "Algorithm for Computer Control of Digital Plotters," *IBM Syst. Journal*, Vol. 4, No. 1, pp. 25-30, 1965.

[5] C. K. Cowan, D. M. Chelberg, and H. S. Lim, "ACRONYM Model Based Vision in the Intelligent Task Automation Project," *First Conference on AIA*, pp. 176-183, 1984.

[6] J. F. Canny, "Finding Edges and Lines in Images," *Master's Thesis, MIT*, 1983.

[7] H. Freeman, "Shape Description Via the Use of Critical Points," *IEEE Conference on Pattern Recognition and Image Processing*, pp. 168-174, June 1977.

[8] T. F. Knoll and R. C. Jain, "Recognizing Partially Visible Objects Using Feature Indexed Hypotheses", *IEEE Journal of Robotics and Automation*, vol. RA-2, No. 1, pp. 3-13, March 1986.

[9] J. Mattill, "The Bin of Parts Problem and the Ice-Box Puzzle," *Technology Review*, Vol. 78, No. 7, pp. 18-19, June 1976.

[10] J. W. McKee and J. K. Aggarwal, "Computer Recognition of Partial Views of Curved Objects," *IEEE Transactions on Computers*, vol. C-26, No. 8, pp. 790-800, August 1977.

[11] W. A. Perkins, "Simplified Model-based Part Locator," *Proceedings of the 5th International Conference on Pattern Recognition*, pp. 260-263, December 1980.

[12] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, New York/ San Francisco/London: Academic Press, pp. 109-123, 1976.

[13] J. L. Turney, T. N. Mudge, and R. A. Volz, "Recognizing Partially Occluded Parts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-7, No. 4, July 1985, pp. 410-421.

[14] J. L. Turney, T. N. Mudge, and R. A. Volz, "Solving the Bin of Parts Problem," *Vision 86 Conference Proceedings*, pp. 4-21 - 4-38, 1986.

[15] J. L. Turney, "Recognition of Partially Occluded Parts", *PhD. dissertation, University of Michigan*, 1986.

[16] K. R. Yam, W. N. Martin, and J. K. Aggarwal, "Analysis of Scenes Containing Several Occluding Curvilinear Objects," University of Texas at Austin Technical Report TR-135, February, 1980.