# 3-D Object Recognition Using Bipartite Matching Embedded in Discrete Relaxation

Whoi-Yul Kim and Avinash C. Kak, *Member, IEEE*

*Abstract*—**In this paper we show how large efficiencies can be achieved in model-based 3-D vision by combining the notions of discrete relaxation and bipartite matching. The computational approach we present is empirically interesting and capable of pruning large segments of search space—an indispensable step when the number of objects in the model library is large and when recognition of complex objects with a large number of surfaces is called for. We use bipartite matching for quick wholesale rejection of inapplicable models. We also use bipartite matching for implementing one of the key steps of discrete relaxation: the determination of compatibility of a scene surface with a potential model surface taking into account relational considerations. While we are able to provide the time complexity function associated with those aspects of the procedure that are implemented via bipartite matching, we are not able to do so for the iterative elements of the discrete relaxation computations. In defense of our claim regarding computational efficiencies of the method presented here, all we can say is that our algorithms do not take more than a couple of iterations even for objects with more than 30 surfaces.**

*Index Terms*— **Artificial intelligence, automatic scene interpretation, bipartite graph matching, computer vision, discrete relaxation, machine intelligence, robot vision, sensor-based robotics, structured-light vision, 3-D vision.**

## I. Introduction

During the past few years, much has been done in the development of strategies for recognition of 3-D objects from range maps, a problem that is fundamentally of exponential complexity unless one discovers and exploits the geometric and topological constraints available. As is true of many such problems—problems which at first sight appear to be NP-complete—the current wisdom for solving the recognition problem dictates the use of some sort of a hypothesize-and-verify approach, with hypothesis formation exploiting the fact that the pose of a rigid object can be estimated rather accurately using a very small number of features, often just two or three, and verification depending on establishing a correspondence between the features actually present in the scene and those predicted by the hypothesized identity and pose of the object. In a recent contribution from our laboratory [5], [6], one such scheme with a computational. complexity of only $O(n^2)$ was presented, where $n$ is the average number of features on a model object. Although this complexity measure is only applicable to the case of single-object recognition, the formalism and the results presented in [5] and [6] were for multiobject scenes containing occlusions. Of course, the contribution that was made in [5] and [6] would not have been possible without some highly noteworthy efforts that preceded it. These previous efforts [3], [10]–[12], [17], [23], [25], [32], [33], [35], [42] uncovered many of the problems in implementing a hypothesize-and-verify approach for 3-D object recognition.

In this paper, our aim is to introduce the reader to a new computational procedure for recognizing objects using range maps. Basic to the computational procedure is the notion of maximal matching in bipartite graphs, a problem for which a low-order polynomial time solution is known [24]. A bipartite graph consists of two sets of nodes and the problem of finding a maximal matching consists of discovering a maximum cardinality set of edges connecting the nodes in the two sets (when the cardinality of a matching equals the cardinality of one of the sets, it is called a complete matching). If we consider the two sets of features, one computed from the given range map and the other drawn from a model object, as corresponding to the two sets of nodes in a bipartite graph, one might be able to perceive a certain similarity between the problem of object recognition and the problem of finding a complete matching in a bipartite graph.

Of course, except for objects with no symmetries whatsoever, it is not possible to directly apply the algorithm of [24] for solving the object recognition problem, because in general there will exist many complete matchings between the nodes corresponding to the scene object and those corresponding to a model object, and because the problem of finding all the complete matchings is in itself NP-complete. It is therefore necessary to embed the bipartite matching algorithm in a larger computational framework, such as the one presented in this paper. We will use the polynomial complexity algorithm for finding complete matchings in bipartite graphs for quick rejection of inapplicable models—since such models will not give rise to a single complete matching with the range data features—and, when an applicable model object is found, use a combination of bipartite matching and discrete relaxation to severely prune the set of possible hypotheses for the pose of the object. Our use of discrete relaxation, although novel from the standpoint of its usage in conjunction with bipartite matching, has a long history of application in various aspects of computer vision [21], [39].

We believe that our scheme has the virtue of fast rejection of models inapplicable to the range map of a given object, the speed advantage owing to the low-order polynomial time complexity of bipartite matching. Therefore, if an application involves a very large number of model objects in the library, and the goal is to recognize an object from its range map, our computational approach would probably work faster than those published hitherto. Of course, this is only a conjecture on our part, not testable at this time because of the difficulty of comparing directly the approaches published by various researchers. In any case, regardless of its computational advantages, the procedure presented in this paper is empirically interesting and therefore

Manuscript received June 22, 1989; revised July 3, 1990. Recommended for acceptance by O. D. Faugeras. This work was supported by the National Science Foundation under Grant CDR8803017 to the Engineering Research Center for Intelligent Manufacturing Systems.

W. Y. Kim was with the Robot Vision Laboratory, Purdue University, West Lafayette, IN 47907. He is now with the School of Engineering and Computer Science, University of Texas, Dallas, TX 75083.

A. C. Kak is with the Robot Vision Laboratory, Purdue University, West Lafayette, IN 47907.

IEEE Log Number 9040359.

worthy of dissemination, if only from the standpoint of its being a respectable alternative to other approaches.

Although, as demonstrated by our experimental results, our computational approach is applicable to multiobject scenes containing occlusions, much of our discussion in this paper will assume, for the sake of simplicity, that only a single object exists in the scene and the goal is to identify the object and compute its pose. For multiobject scenes, our system works under the constraint that interobject boundaries be characterizable as jump edges in the range data. In other words, if objects are juxtaposed in such a manner that surfaces from different objects come together to form surface transitions that are $C^0$ continuous, then our approach will fail to recognize such objects. For example, in accordance with the results discussed in Section VI-B, for the scene in Fig. 1(a) our approach will correctly identify objects $G_2$, $G_3$ and $G_4$ but will fail for object $G_1$ (the labels for the objects and the surfaces are shown in the preprocessed range map in Fig. 1(c) while Fig. 1(b) shows a light stripe image of the scene). As with other such objects, successful recognition and pose estimation for object $G_2$ results in the kind of manipulation shown by the sequence of frames in Fig. 2, where we show a robot picking up the object using an approach angle and gripper attitude that is consistent with the computed pose transform for $G_2$. The failure to recognize object $G_1$ in Fig. 1(c) is caused by the fact that two of its features, surfaces 1 and 28, are not separated by range discontinuity edges from surfaces 5 and 26, respectively, of the adjoining objects.[1] We also use CAD-generated images, like the one shown in Fig. 1(d), to illustrate the recognition of objects and calculation of their poses. The object images in Fig. 1(d) were generated by invoking a solid modeling package with the identities and the computed pose transforms for the objects $G_2$, $G_3$, and $G_4$ in Fig. 1(a).

In Section II, we review past work relevant to the current exposition. We then describe in Section III the object features that are used in our scheme and their various attributes; this section also contains discussion on why the technique of bipartite matching is relevant to the problem of object recognition. Section IV defines more formally the notions of scene and model graphs and presents criteria for matching a feature from the scene, the feature being a node in the scene graph, with a feature from a model, this latter feature being a node from a model graph. Section IV also presents criteria for matching relations from scene and model graphs. We then discuss the overall flow of control in Section V, where we present a serialized architecture consisting of various modules that invoke different considerations to reject inapplicable candidate models. The modules have been arranged in such a manner that strategies that are computationally easy, but yet powerful for eliminating inapplicable models on the basis of more global properties, are first invoked. Finally, in Section VI we show experimental results on single- and multiobject scenes.

## II. Past Work

Choice of features, representation of objects on the basis of features, and the organization of matchings between the scene and the model features are the key issues in solving the problem of object recognition and pose estimation. Evidently, the features used should be such that they can be extracted reliably in the presence of noise and other distortions common to range sensing; features should also lend themselves to easy computation even when the models are mathematically specified by the equations of their surfaces. After features are specified, one must address the related issues of how to represent objects using these features and how to organize the search associated with matching scene features with object features.

In the past, researchers have used some combinations of surface, edge and point features, although in some cases only a single type was used. For example, Oshima and Shirai [36] and Fan *et al.* [9] have used surfaces and their geometric and topological relationships. In the case of Oshima and Shirai, surface-based features were organized into graphs whose nodes were the surfaces and whose links the relationships between the surfaces, the type of relationships being adjacency, convexity or concavity of common edges, dihedral angles, distance between the centroids, etc. In the work of Fan *et al,* the links in a graph representation whose nodes correspond to surfaces are given weights depending upon the nature of edges between the surfaces. Surface-based features have also been used by Bhanu [2], although the notion of adjacency between surfaces here is radically different from what is used in [9] and [36]; this is owing to the fact that in [2] each curved surface is broken into its planar approximations. As opposed to surfaces, Umeyama *et al.* [42] have used edges for recognizing objects from 3-D data. Another effort along similar lines is that of Hebert and Kanade [23]; they also have used edges.

Amongst those who have used a combination of different feature types, Bolles and Horaud [3], Faugeras and Herbert [12], Grimson and Lozano-Perez [15]–[18], and Ikeuchi [26] come to mind. Bolles and Horaud have used edges to generate hypotheses about object identity and pose, and then used surfaces for verification. The matching scheme of Bolles and Horaud starts from a distinctive scene feature that is matched to its corresponding model feature; a pose hypothesis is subsequently generated by seeking matches for a small number of scene features that are adjacent to the first extracted distinctive feature. Verification consists of projecting into the scene a synthetically generated range map of the object for the hypothesized pose and comparing this range map with actual range data. As matches are scored between the two, the pose hypothesis is upgraded continually. Faugeras and Hebert [12] use prominent edges and large surfaces for hypothesis formation; verification is carried out by keeping track of a registration error computed initially during the hypothesis formation stage and updated subsequently when model to scene matches are attempted for those model features that should be visible in the scene for the hypothesized pose. Grimson and Lozano-Perez have used object points, surfaces, and surface normals in an approach which consists of associating with a given set of measurements only those objects and poses that satisfy constraints with regard to the measurements. Of course, since local constraints are not sufficient to fully specify the pose of an object, especially when the identity of the object is also unknown, they subsequently compute an average of all the possible pose transforms obtained from the local constraints and see if under this average transform a predicted model can give rise to the measured points. More recently, the authors have extended their technique to objects with "parameterizable shapes," as would, for example, be the case with scissors when the angle between the blades, which is allowed to be variable, is used to serve as a parameter of shape. In the scheme proposed by Ikeuchi [26], all possible object poses are first grouped

[1] This shortcoming should not prove fatal if our system were to be employed for bin picking in a manufacturing operation. After removing the recognized objects, the bin could be mechanically disturbed, either by a robot or a shaker, and the scene rescanned. It is rather unlikely that two surfaces from different objects that, owing to the nature of their juxtaposition, originally formed a single $C^0$ continuous surface would continue to do so after the objects are mechanically disturbed.

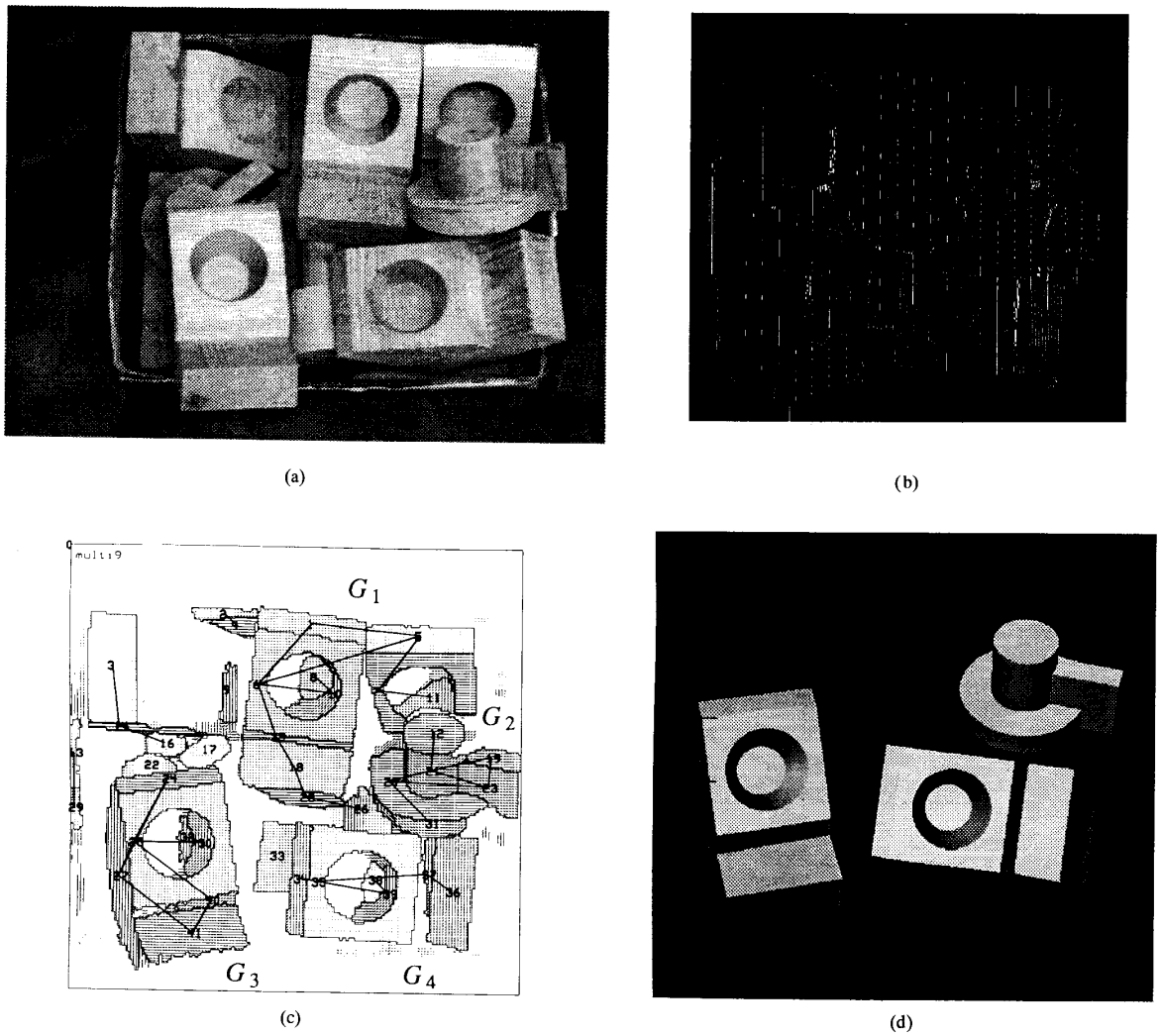



Fig. 1. (a) A pile of nonpolyhedral objects. (b) A light stripe image of the scene generated by a robot hand-held structured light scanner. (c) Preprocessed range image obtained from the light stripe data. Preprocessing involves segmentation, surface classification and generation of a scene graph. (d) Recognition of some of the objects and calculation of their poses is illustrated here by invoking a solid modeling package with the object identities and the computed pose transforms.

into "attitude groups," each attitude grouping being a set of aspects on the basis of commonality of prominent surfaces that are visible; each aspect is a collection of those poses that are topologically equivalent, in the sense that from a given viewpoint the same object surfaces are visible. Ikeuchi then constructs an interpretation tree in which each node corresponds to a presence or an absence of prominent surface. As one discovers neighboring surfaces in the scene, one can go down the interpretation tree and try to categorize the pose of the object as belonging to one of the attitude groups. Eventually, separate strategies are invoked to calculate more precisely the object pose from its attitude group. While attitude-grouping calculations use regions extracted from depth maps constructed by photometric stereo, the more refined pose calculations are carried out using edge maps, extended Gaussian images, etc.

In Jain and Hoffman's work [28], a range map is segmented into three types of regions corresponding to convex, concave, and planar surface types. This initial segmentation is followed by a merging operation based on knowledge of the extent to which different surfaces can be allowed to differ in their orientations at their common boundaries. Various attributes of these merged surface patches are then measured using a rule-based framework for a statistical classification of the object. While some attributes directly characterize a merged patch, for example, attributes like area, surface type, etc., other attributes are more relational and global in nature, examples of the latter type being the angle between two surface normals, the perimeter of the object silhouette, etc. For the rule-based implementation, the rules are generated automatically from the interobject distances in the multidimensional attribute space. Wong *et al.* [43]–[45] have used a hierarchical graph representation scheme in which an object is considered to be made of more primitive blocks, each block possessing its own graph representation. The object-level representation is called a hypergraph, which is basically a grouping imposed on the vertices of a more detailed graph, each vertex of the more detailed graph corresponding to a surface of

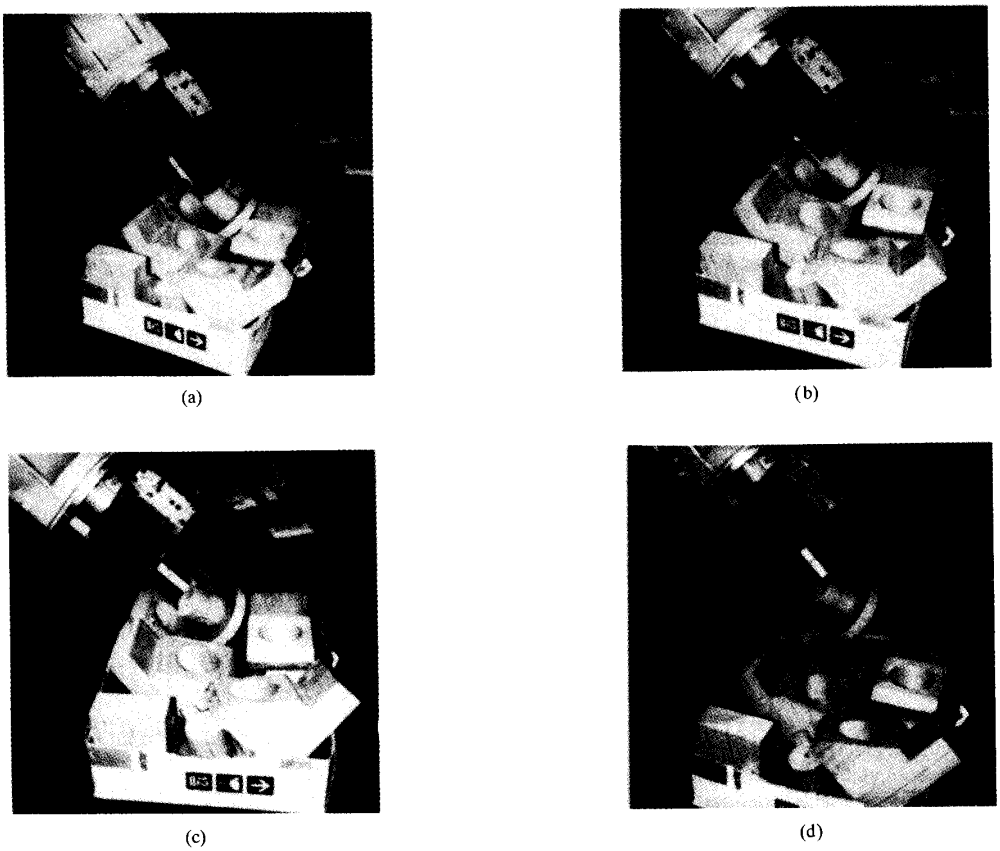


Fig. 2. Sequence of frames showing a robot picking up one of the objects that was successfully recognized from the multiobject scene shown in Fig. 1(a).

the object and each group in the grouping corresponding to a primitive block of the object. Such a hierarchical representation leads to a reduction in the complexity of matching, since the recognition of primitive blocks can be used to quickly eliminate inapplicable model objects.

Another recent effort that used a combination of surface and vertex features is from our laboratory [5], [6]. This work, whose effectiveness was demonstrated experimentally on multiobject scenes containing occlusions, resulted in a scheme for recognition and pose estimation whose time complexity is only $O(n^2)$ for single-object scenes. In this scheme, each feature type is arranged in a spherical data structure in a manner that eliminates virtually all search needed for the verification of a pose/identity hypothesis. In our opinion, this work constitutes a demonstration of the fact that the complexity associated with object recognition is greatly influenced by the representation used for objects.

The organization of search in all the methods cited above may be considered to be either model driven or data driven. Data-driven searches tend to be more efficient because the number of object features visible from any viewpoint will be less than the total number of features on the object. Therefore, given an object hypothesis, it is more efficient to match scene features to model features than vice versa. Nonetheless, in some cases it may be advantageous to use a model-driven approach, especially when all the objects in a pile are identical, in which case efficiencies in both hypothesis formation and verification can be had by recognizing the fact that a scene feature must be one of the small number of features on the model object. For example, in the approach used by Faugeras and Herbert [12], model features are first ranked according to some attribute; the feature with the highest ranking is then sought in the scene. After a match is scored with a model feature, a search is conducted in the scene for another model feature under the constraint that the geometrical relationship between the two model features be identical to the two corresponding scene features. Clearly, if this model-driven strategy were to be extended to scenes containing objects of different types, it would be inefficient due to the excessively large search spaces involved. Model-driven approaches also appear to be superior for recognizing objects that are occluded "in the middle." In a model-driven approach, one of the visible parts could be used for generating a pose hypothesis. The model objects could then be transformed into the scene using the pose transform corresponding to the hypothesis and the correspondence registered with the visible fragments of the scene object could be used as a measure of verification.

## III. Features, Attributes, and Organization of Search

The problem of object recognition may be formally stated as follows. Given a scene object $S$ and a library of model objects as represented by the set $M_1, M_2, \cdots$, we wish our recognition

system to tell us that[2]

$$S_i \Rightarrow (M_j, \boldsymbol{T})$$

meaning that the scene object $S_i$ is an instance of the model object $M_j$ and that the pose transform $T$ takes the scene object into the model object. In Fig. 3(a) and (b), we have shown the features associated with a scene object in a random pose and the corresponding model object, the features being surfaces and edges. In the computer memory, each feature is represented by a frame data structure, meaning a list of attribute-value pairs. For the objects shown in Fig. 3(a) and (b), the attribute frames for the scene feature 4 and the model feature *b* are shown in Fig. 3(c) and (d), respectively.

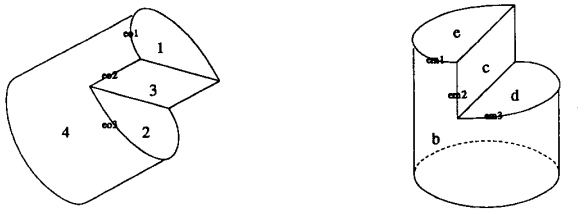
That the problem of object recognition is related to the problem of bipartite matching may be seen with the help of Fig. 4(a) where the left column displays the set of scene features, $s_1, s_2, \cdots$ and the right column the set of model features $m_1, m_2 \cdots$. Considered by itself, a scene feature may bear similarities with many model features, as illustrated by the arcs in Fig. 4(b). However, for recognition to be correct, we want every scene feature to be matched to a distinct model feature and no two model features to be matched to the same scene feature. Fig. 4(c) shows a possible matching that would be acceptable. The task then becomes to extract from all the feature matchings shown in Fig. 4(b) those that are injective, meaning that they satisfy the properties of one–one and into.

A graph is called bipartite if all the nodes can be divided into two disjoint groups, as, for example, in Fig. 4(a), and if each arc in the graph connects two nodes belonging to the two separate groups. By *bipartite matching* we mean assigning to each node from one of the groupings a node from the other grouping. For example, the problem of finding arcs from Fig. 4(c) such that each scene node is connected to a model node would be an exercise in matching. In general, it is not necessary that every scene node be assigned a model node, but when that is the case, we have a *complete matching*. In other words, the cardinality of a complete matching is equal to the cardinality of one of the two node groupings in a bipartite graph.

Evidently, our object recognition problem bears great similarity to the problem of bipartite matching; however, there is one caveat, namely, that while a complete matching is necessary for a matching to constitute object recognition, it is not sufficient. In other words, just because a matching between the scene nodes and the model nodes is complete, meaning that we have found a distinct model feature to match with every scene feature, that is not sufficient reason to declare a match at the object level. Bipartite matching by itself is incapable of imposing relational constraints that must be satisfied by both the scene features and the model features. Without these relational constraints, and especially when the objects involved are symmetric, implying that they contain features that have identical attributes, it would only be too easy to assign model features to scene features in a manner that may not correspond to the existence of any rigid-body transform between the scene object and the model object.[3]

[2]In accordance with what was said in the Introduction, this problem statement assumes that only a single object is present in the scene. For the case of scenes containing multiple objects, the problem statement applies when a collection of surfaces, presumed to belong to a single object, is being analyzed for object identification and pose estimation.

[3]At this point, the reader is probably thinking that despite this connection we have drawn between object recognition and bipartite matching, fundamentally we are still facing the problem of subgraph isomorphism. However, the reader should note that invoking the notion of bipartite matching has a major advantage, viz., the existence of a complete matching can be carried out via a polynomial time algorithm whereas subgraph isomorphism still remains exponentially bound. Later, we will have more to say about this efficient algorithm and how we have used it in our system. The reader should also note one previous effort at using the notion of bipartite matching in the context of scene analysis [41].




{ feature-index : 4
<surface-type : cylindrical>
<area : 4.5>
<adj-list : (eo1 1) (eo2 3) (eo3 2)>
<radius : 1.0>
<height : 3.5>
}

(c)

{ feature-index : b
<surface-type : cylindrical>
<area : 4.8>
<adj-list : (em1 e) (em2 c) (em3 d) (em4 a)>
<radius : 1.1>
<height : 3.6>
}

(d)

Fig. 3. Feature frames associated with a surface in a scene object and a model object. (a) A scene object in a random pose. (b) The corresponding model object in its standard pose. (c) Feature frame for feature 4 of the scene object. (d) Feature frame for feature *b* of the model object.

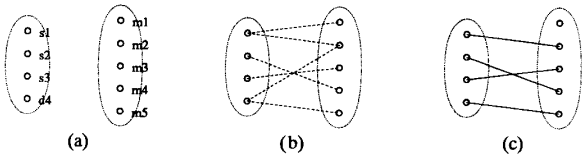



Fig. 4. (a) The left column displays a set of scene features and the right column a set of model features. (b) If an arc joins a scene node with a model node, that means the two nodes are similar in some sense. (c) As illustrated here, an acceptable matching between scene nodes and model nodes must be injective.

Therefore, we must extract from all possible complete matchings those that correspond to feasible transformations between the scene and the model objects [27]. This is an ideal problem for search and we have chosen to implement it via discrete relaxation.

There is a bit more to our implementation than a straightforward extraction of a complete matching via discrete relaxation. As will be discussed in Section V, we use bipartite matching in two separate phases. Initially, bipartite matching is used to quickly establish the existence of at least one complete matching, since the absence of one means that the model is not applicable. Subsequently, bipartite matching is also used to fine-tune feature correspondences by insisting that the set of features in the vicinity of a given feature also possesses a complete match with a corresponding neighboring set of features for the model feature in question. For a given feature in the scene or the model, the neighboring features for this purpose are selected on the basis of relational attributes, such as adjacency.

We would now like to discuss some of the distinctions we make with regard to various attributes. An attribute associated

with a surface feature may be either *intrinsic* or *extrinsic;* the attribute is intrinsic if its values are independent of the pose and location of the object, otherwise the attribute is extrinsic. This distinction is important because the attributes that are used for hypothesizing the pose of an object should be independent of the pose parameters. For example, we may use the attribute *area* for matching a scene surface with a model surface during hypothesis formation stage, but not the *normals* associated with the two surfaces. We could not use the latter attribute even if we wanted to because we do not yet have a pose transform for the object and therefore we really know nothing about the direction of the surface normal for the scene surface. In Table I, we have listed the intrinsic and extrinsic attributes for different feature types. While some of the attributes are clear from the names used, a couple need further explanation. For the feature type "cylindrical," the attribute *+radius* is instantiated to the value of the radius if the cylindrical surface is convex, like the outside of a cup. On the other hand, if the cylindrical surface is concave, like the inside of a cup, the attribute *−radius* is instantiated to the value of the radius. Distinctions between convex and concave cylindrical surfaces are obviously important to recognition. The attributes *+base_radius* and *−base_radius* are to be interpreted in a similar manner.

We have captured one more distinction between attributes in Table I, namely between attributes that are viewpoint independent and those that are viewpoint dependent. Consider, for example, the attribute *area.* As shown by a perspective view in Fig. 5, the feature *s*2 is fully visible and therefore its area attribute can be computed accurately. On the other hand, the computed area attribute of the feature *s*3 will be less than the true area of that feature due to self-occlusion. Clearly, the attribute *area* is viewpoint dependent. In comparison, the attribute *radius* associated with a cylindrical surface is viewpoint independent. Of course, for any viewpoint, we would need a large enough patch of the cylindrical surface for estimating its radius, yet the nature of this dependence is not the same and not as direct as the dependence of the area of a planar surface as shown in Fig. 5. In Table I, attributes that are viewpoint dependent are marked by the suffix @. When we compare such an attribute measured in a scene with its corresponding value from a model, we cannot insist upon equalities; the most we can hope for is for the scene value to be less than the model value.

As was mentioned before, in addition to surface features our system also uses edge features. As will be clear from the next section, surface features constitute the nodes of an attributed graph representation for objects; the arcs of this graph correspond to edge features. Just like surfaces, edge features have attributes, too. However, in contrast with surfaces, edge features only have intrinsic attributes, and these are shown in Table II. Each edge can only have two attributes: *edge_type* and *dihedral_angle.* The allowed *edge-types* are convex, concave, inflected, and cavex[4]; the last two types reserved for edges of the types in models 4 and 6 are shown in Fig. 6. Inflected edges are characterized by $C^2$ discontinuity across them, as is the case with edge *EF* in model 4 in Fig. 6. An example of a cavex edge is shown in model 6 in Fig. 6. Since part *B* of the edge is convex while part *A* is concave, this edge is defined as cavex.[5] As the reader might be able to tell from the table, we have taken some liberty with the conventional usage of the term *dihedral_angle.* The term usually stands for the angle between two planar surfaces at the edge where they meet. Evidently, this angle must be defined for junctions between surfaces where one or both of the surfaces are nonplanar. The information below the table displays the definition of this angle for each case. Note that in the table, entries like "(planar, planar)" are not values themselves for the dihedral angle attribute; these

TABLE I
FEATURES TYPES AND THEIR ATTRIBUTES

| Feature Type | intrinsic attributes | extrinsic attributes |
|---|---|---|
| planar | *area@*<br>*adjacency_list@* | *normal*<br>*centroid@* |
| cylindrical | *+radius*<br>*−radius*<br>*height@*<br>*adjacency_list@* | *axis*<br>*centroid@* |
| spherical | *+radius*<br>*−radius*<br>*adjacency_list@* | *center* |
| conical | *+base_radius*<br>*−base_radius*<br>*+top_radius*<br>*−top_radius*<br>*slant_angleθ*<br>*height@*<br>*adjacency_list@* | *axis*<br><br><br>*peak_location* |
| other quadric | *+eigenvalues*<br>*−eigenvalues*<br>*adjacency_list@* | *eigenvectors*<br>*center* |

*adjacency_list* consists of number and list of adjacent surfaces, i.e., $(|\Gamma(x)|, \Gamma(x))$.
Symbol "@" denotes viewpoint dependent attribute.
Symbol "−" denotes peak surface.
Symbol "+" denotes pit surface.
For cylinders of split height, the value of *height* is the maximum height.

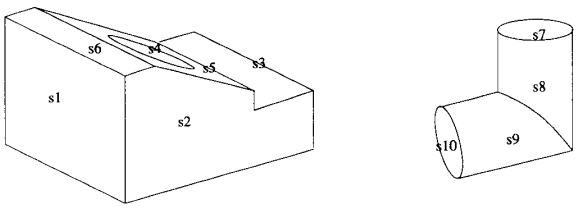



Fig. 5. To cope with occlusions, a distinction must be made between attributes that are viewpoint independent and those that are not. For example, the *area* attribute is viewpoint dependent, since its value is subject to occlusion, as exemplified by the feature *s*3 of the object on the left. Other attributes, such as the *radius* of the feature *s*8 for the object on the right, are considered to be viewpoint independent.

[4] The term "cavex" stands for a combination of convex and concave. These edges between smooth surfaces are partly convex and partly concave. Most often, such edges are created by joining two cylinders at an angle. For example, the edge between the surfaces *s*8 and *s*9 in Fig. 5 is partly concave and partly convex.

[5] Edges in 3-D scenes may be characterized by the nature of discontinuities. A jump edge is characterized by $C^0$ discontinuity, a roof edge (same thing as a convex edge), and a crest edge (same thing as a concave edge) by a $C^1$ discontinuity. Then we have edges with smooth transitions between relatively flat surfaces, as exemplified by model 4 in Fig. 6. In addition, we can have occluding and occluded edge_types in range maps; these are not used in our system because they really are not physical features and are dependent on the parameters of the sensor, in addition to, of course, the viewpoint.

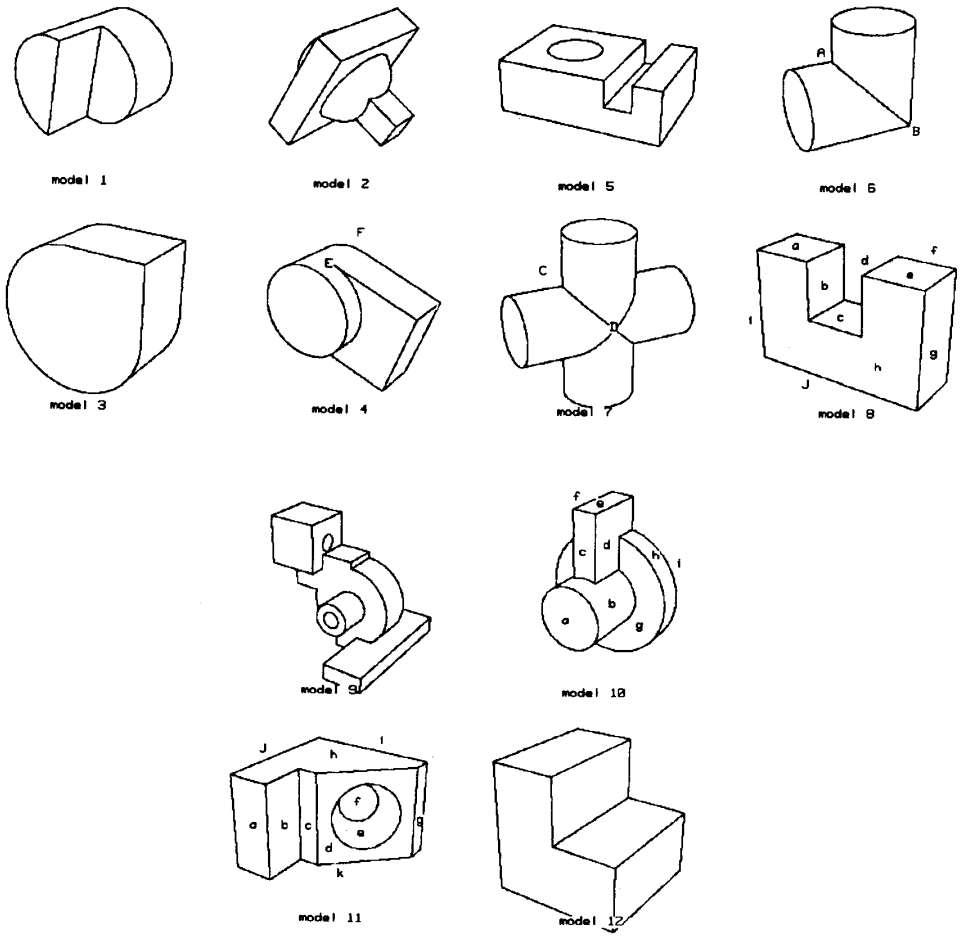



Fig. 6. A set of CAD models used in our experiments.

entries merely show which of the definitions presented below the table must be used for the angle for each case.

## IV. Representation of Objects and Matching Criteria

An object is represented by an attributed graph whose nodes are surface features and whose arcs are the edges between the surfaces. Fig. 7(c) and (d) shows examples of such representations for the scene and the model objects in Fig. 7(a) and (b), respectively. The problem of object recognition is then to start with a scene object representation, such as the one shown in Fig. 7(c), compare this representation with all the objects in the model library, and arrive at possible interpretations. We will use $M = (N_M, A_M)$ to represent a model graph; $M$ will be subscripted suitably when more than one model graph is involved. We will use $S = (N_S, A_S)$ to represent a scene graph. $N_M$ and $N_S$ are the sets of nodes in the two graphs, and $A_M$ and $A_S$ the sets of arcs. In terms of the surface features present, each denoted by $s_i$, the set of nodes $N_S$ will be described by

$$N_S = \{s_i | i = 1, \cdots, |N_S|\}.$$

Similarly, the set of nodes in a model graph may be expressed as

$$N_M = \{m_i | i = 1, \cdots, |N_M|\}$$

where each $m_i$ is a surface feature on the model.

Any recognition strategy must of necessity compare model and scene features via their attributes and measure their consistency with respect to one another in the sense defined below. One advantage of categorizing attributes in the manner we did in the preceding section is that different criteria can be used for different types of attributes. We could insist on equalities, within of course the limits imposed by measurement noise, when the system compares attributes that are viewpoint independent; however, a viewpoint-dependent scene attribute should be allowed to take any value that is less, in a numerical or set-theoretic sense, than that of its counterpart in the model.

More formally, we will define three types of consistencies as follows.[6]

### A. Node Consistency

Let $f_a$ be an operator that when applied to a feature returns the value of the attribute $a$. First consider the case when $a$ is a numerical attribute. We consider a scene feature $s_j$ *consistent*

[6]The last type of consistency will actually be called compatibility in keeping with the conventional usage in the discrete relaxation literature.

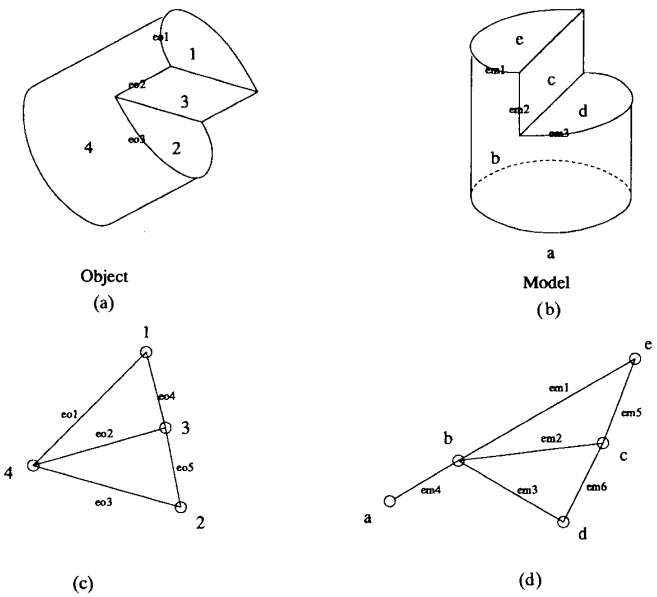



Fig. 7. (a) A scene object in a random pose. (b) The corresponding model object. (c) Graph representation of the scene object. (d) Graph representation of the model object.

TABLE II
ARC ATTRIBUTES

| attribute | value | |
|---|---|---|
| *edge_type* | convex | |
| | concave | |
| | inflected | |
| | cavex | |
| *dihedral_angle* | $\theta_1$ | (planar, planar) |
| | $\theta_2$ | (planar, cylindrical) |
| | $\theta_2$ | (planar, conical) |
| | $\theta_3$ | (cylindrical, cylindrical) |
| | $\theta_3$ | (cylindrical, conical) |
| | $\theta_3$ | (conical, conical) |
| | N/D | (other combinations) |

$\theta_1$: angle between two surface normals
$\theta_2$: angle between axis and surface normal
$\theta_3$: angle between two axes
N/D: Not Defined

with a model feature $m_i$ if

$$f_a(s_j) = f_a(m_i)$$

for a viewpoint-independent $a$, or if

$$f_a(s_j) \leq f_a(m_i)$$

when $a$ is viewpoint dependent; these relationships must hold for all the numerical attributes defined for the model feature $m_i$. As mentioned before, for practical implementation the equality condition would only be enforced within the limits set by measurement noise. When an attribute is symbolic, a scene feature $s_j$ is consistent with a model feature $m_i$ if

$$|f_a(s_j)| \leq |f_a(m_i)|$$

where $|\cdot|$ represents the cardinality of its argument. Again, as with numerical attributes, these relationships must hold for all the symbolic attributes defined for $m_i$. As the reader will note from Table I, *adjacency_list* is the only symbolic attribute and it happens to be viewpoint dependent. The consistency criterion shown above is appropriate for viewpoint-dependent symbolic attributes.

When a node $s_j$ in a scene graph is node-consistent with a node $m_i$ in a model graph, that fact will be denoted by $s_j \rightarrow m_i$.

### B. Arc Consistency

Let $s_{i,k}$ and $m_{j,l}$ represent, respectively, arcs in $S$ and $M$. Arc $s_{i,k}$ will be considered to be consistent with arc $m_{j,l}$ if the scene node $s_i$ is consistent with the model node $m_j$, and $s_k$ with $m_l$; in addition, the following conditions must be satisfied:

$$f_{edge\_type}(s_{i,k}) = f_{edge\_type}(m_{j,l})$$
$$f_{dihedral\_angle}(s_{i,k}) = f_{dihedral\_angle}(m_{j,l}).$$

In the first equation, when one of the edge types involved is cavex in the sense discussed in the previous subsection and as exemplified by edge *AB* of model 6 in Fig. 6, the equality must be interpreted somewhat loosely. When the *edge type* of a model edge is cavex, either convex, concave, or cavex scene edges are allowed to match with that. When an arc $s_{i,k}$ in a scene graph is arc-consistent with an are $m_{j,l}$ in a model graph, we denote that fact by $s_{i,k} \rightarrow m_{j,l}$.

### C. Local Compatibility

Readers familiar with relaxation methods will recall that such methods need a measure of compatibility between two different class assignments. To elaborate, suppose we want to classify a set of entities $E_1, \cdots, E_n$ into $m$ classes $C_1, \cdots, C_m$ by using relaxation. We need to define a compatibility measure $c(i, j; k, l)$ which is a measure of how compatible the assignment of object $E_i$

to class $C_j$ is with the assignment of object $E_k$ to class $C_l$. Local compatibility defined here is a generalization of $c(i,j;k,l)$, in the sense that we now consider more than two class assignments at a time.

More formally, a node $s_p$ in a scene graph will be considered to be *locally compatible* with a node $m_q$ in a model graph if the following condition is satisfied:

$$\forall_{1\le k\le|\Gamma(s_p)|}\exists_{1\le l\le|\Gamma(m_q)|}\{s_p \to m_q, s_{p,k} \to m_{q,l}, s_k \to m_l\}$$

where $\Gamma(x)$ is the set of nodes adjacent to node $x$. We require that the mapping implied by $s_{p,k} \to m_{q,l}$, be injective, meaning there be a one–one and into mapping between the arcs incident at node $s_p$ in the scene graph and the arcs incident at node $m_q$ in the model graph, while we allow the degree of $m_q$ to be equal or larger than the degree of $s_p$. Note that when $s_{p,k} \to m_{q,l}$ leads to an injective mapping between arcs in the neighborhoods around $s_p$ in the scene graph and $m_q$ in the model graph, there will also be an injective mapping between the nodes in these neighborhoods.

As is evident from our definition, a scene node is locally compatible with a model node provided there exists node consistency between the two and, also, provided there exists node consistency between the potentially corresponding nodes from the immediate neighborhoods of the two; in addition, the respective arcs in the immediate neighborhoods must also satisfy arc consistency. That our definition of local compatibility could be considered a generalization of the function $c(i,j;k,l)$ should be apparent if the reader realizes that for our problem the compatibility $c$ could be made proportional to the similarity of, say, the angle between the surfaces $i$ and $k$ in the scene and the angle between the surfaces $j$ and $l$ in the model, assuming for the sake of argument that all the surfaces are planar. In our definition of local compatibility, relationships of this type are captured by the arcs $s_{p,k}$ and $m_{q,l}$.

## V. Flow of Control

### A. Overview

Given a scene graph and a library of model graphs, the overall control of the system, as illustrated in Fig. 8, is composed of three modules, namely, the filtering module, the pruning module, and the ambiguity resolution module. The filtering module, which carries out a fast rejection of inapplicable models, consists of three filters, arranged in the order of increasing computational complexity. The candidate models at the output of the filtering module are then processed by the pruning module, which invokes a combination of bipartite matching and discrete relaxation to reduce the size of the search space for the formation of hypotheses and their verification. As we will illustrate through experimental results, the pruning process is severe and usually results in comparing, for both hypothesis formation and verification, a scene feature with a very small subset of the total number of features on a model. Finally, the ambiguity resolution module determines the object identity and the pose as well as the location of the object. In the following subsections, we describe each module in detail.

### B. The Filtering Module

The fact that the three filters in this module are arranged in order of increasing computational complexity, as shown in Fig. 8, makes intuitive sense because we want the least amount of work to be done to discard a potential candidate model if it is not applicable.

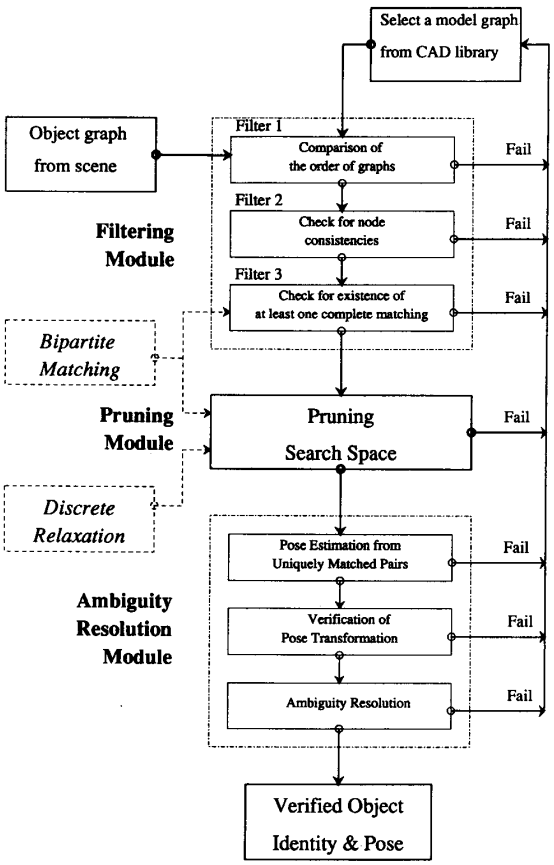



Fig. 8. The flow of control for object recognition and pose estimation. For objects with high degrees of symmetry, the ambiguity resolution module is replaced by the one shown in Fig. 16.

*Filter 1:* The first filter compares the scene graph with a model graph on the basis of the following global properties:

1) The total number of nodes in each graph. The number of nodes in the scene graph cannot be greater than the number of nodes in a candidate model graph for the recognition of a single isolated object.
2) The total number of arcs. As with the nodes, for the recognition of a single isolated object the total number of scene arcs cannot exceed the number in a candidate model graph.
3) The maximum degree of a node. As before, the maximum degree of a scene node cannot exceed the maximum degree associated with any node in a candidate model graph.
4) The first filter also counts the total number of surfaces of different types in each graph. For each type of surface, the count in the scene graph cannot exceed that in a candidate model graph.

The main virtue of the above criteria is that they can be computed very quickly for each graph. However, the ease of computing extracts a price: an inapplicable model graph would not necessarily be rejected by this filter. Consider, for example, the case when a model $M_1$ has four planar surfaces of areas equal to 10, 10, 10, and 20, and a scene object $S$ has two planar surfaces of areas 15 and 15. In this case, while one of the scene surfaces

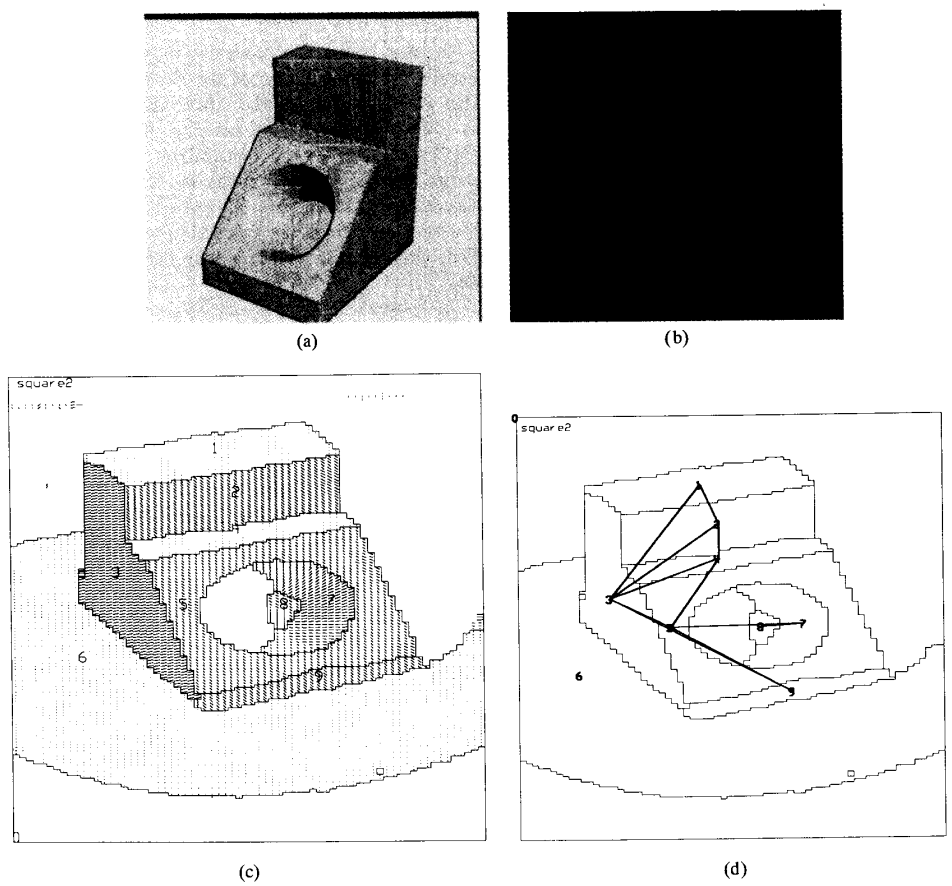



Fig. 9. The range data segmentation procedure used in our system is capable of pulling out surfaces that are at almost the same level of connectivity as the surfaces in the models. (a) A scene object with convex and concave surfaces. (b) A light stripe image of the object. (c) Segmentation of the range map. (d) Graph representation built from the output of the segmentation program.

of area 15 could be matched with the model surface of area 20, the other scene surface would then remain unmatchable, implying the inapplicability of this model. However, the first filter would not be able to reject the model using any of the criteria listed above. Note particularly that the last criterion compares the two graphs on the basis of only the number of surfaces involved and not on the basis of what's needed here—the area attribute. In fact, the first filter makes no discriminations on the basis of any viewpoint-dependent attributes, the area attribute being one of those.

The reader might say that despite the only $O(|N_S|)$ time complexity of the first filter, its power for rejecting a candidate model graph is rather limited owing to the fact that when we are trying to recognize an object from a single viewpoint the scene graph will only be a "partial" graph in comparison with the model graphs and not very rejectable by the above criteria. That is not entirely correct, especially when the library of models is large and varied. The filter will quickly reject "simple" models for "complex" objects in a scene. The filter would become even more effective were we to try object recognition from scene graphs constructed by combining features obtained from multiple viewpoints, since now the scene graph would be more akin to the correct model graph and more different, in terms of the global properties mentioned above, from other model graphs.

The criteria expressed above in terms of global properties of graphs are predicated upon the expectation that the segmentor is capable of pulling out, more or less intact, the visible surfaces of an object. In other words, there is the expectation that the segmenter is not breaking each analytically continuous surface into too many subsurfaces. Given the range maps that can be produced today, it is not too difficult to design such segmentors [19]. For example, Fig. 9(a) shows a fairly complex object, Fig. 9(b) its range map using a composite of light stripes, Fig. 9(c) its segmentation using an approach described more fully in [6], and Fig. 9(d) its graph representation.

*Filter 2:* The purpose of the second filter is to check that every node $s_p$ in the scene graph is in node consistency with some node $m_q$ in the model graph. The reader might initially construe the requirement of the second filter to imply that the segmentation of a range map must be nearly perfect. However, that is not the case. As was stated in our definition of node consistency, when we compare an attribute of a scene node with the same attribute of a model node, we do not always insist that the two attribute values be equal. While allowing for self-occlusion, this also makes the system accommodating of some of the segmentation artifacts. So, if a surface was not segmented out completely, and thus had a reduced area, its node in the scene graph would still be found to be consistent

| | | p | c | p | p | p |
|---|---|---|---|---|---|---|
| | | a | b | c | d | e |
| p | 1 | 1 | 0 | 1 | 1 | 1 |
| p | 2 | 1 | 0 | 1 | 1 | 1 |
| p | 3 | 1 | 0 | 1 | 1 | 1 |
| c | 4 | 0 | 1 | 0 | 0 | 0 |

Fig. 10. An entry of "1" in the assignment-consistency table implies that the corresponding scene and model surfaces are in node consistency. An entry of "0" means absence of node consistency.

with the corresponding node in the model graph. Of course, any small surface fragments generated by the segmentor could simply be discarded at the time the scene graph is made. This elimination process would, of course, cause the system to be unable to discriminate between objects that are alike except for some "small" surface features, but then that is the price one has to pay for dealing with the imperfections of a segmentor. We believe that this tradeoff between sensitivity to small features and tolerance of segmentation artifacts is common to all 3-D vision systems.

The main purpose of the second filter is to discard any models that do not have correspondents for all of the scene features. This filter can be implemented by exhaustively comparing all scene features against all model features, the complexity of this comparison being $O(|N_S| \times |N_M|)$. The result of such a comparison is stored in the form of a table, to be referred to as the *assignment-consistency table;* this table is of size $|N_S| \times |N_M|$, where $N_S$ and $N_M$ are, respectively, the number of nodes in the scene object and the model. To illustrate, for the scene and the model graphs shown in Fig. 7, the assignment-consistency table is shown in Fig. 10. The row labels 1, 2, 3, and 4 correspond to the various visible surfaces in the scene object, while the column labels $a, b, \cdots, e$ designate all the surfaces in the model. The entries in the table are 1 and 0 depending upon whether or not the scene and the model surfaces are found to be in node consistency. In this particular example, the inconsistencies, corresponding to 0's in the table, are all due to the surface types being different. Note that for the convenience of the reader the surface types have also been shown, in the form of *p* and *c* labels, for planar and cylindrical, in the table.

*Filter 3:* To all those models that survive the previous two filters, we now apply a more computationally expensive filtering operation: the bipartite matching operation. The purpose of bipartite matching here is to tell us whether or not there exists a one–one injective mapping between the scene object and a candidate model object. We look for a one–one mapping because we want to make sure that for every scene feature, there exists at least one model feature, and the desired mapping should be injective because not all the model features will be visible in the scene. An important point to note here is that when the features involved are not all distinct, there may exist more than a single one–one injective mapping from the scene object to a candidate model object; the third filter guarantees us that there will be at least one.

In Appendix A, we have presented our implementation of a polynomial time bipartite matching algorithm. The bipartite graph is input to this algorithm in the form of the assignment-consistency table produced by Filter 2. The filter reports either yes or no about the existence of a one–one injective mapping. All the candidate models for which such a mapping is not found are rejected at this time.

### C. The Pruning Module

In most cases, only a small number of model objects will survive all three filters of the filtering module. Those models that do survive constitute the remaining search space, which is further pruned by the pruning module, shown in Fig. 8, on the basis of relational constraints between the features in the scene and the features in the models. We will now show how discrete relaxation, in combination with another application of bipartite matching, can be used to enforce relational constraints. But, first, a few words about discrete relaxation are in order.

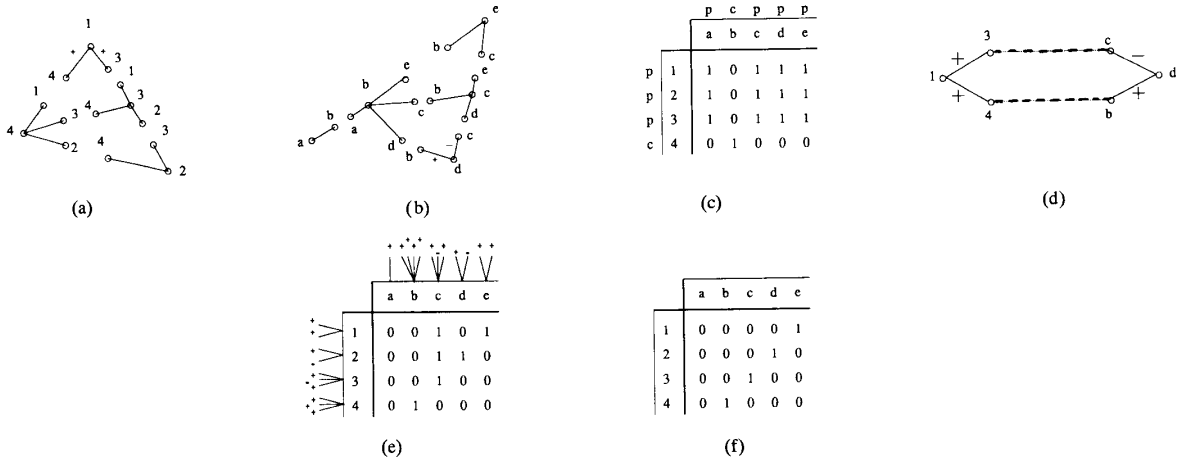
In general, relaxation, as originally proposed by Rosenfeld and Hummel [39] in the computer vision context, refers to a manner of iterative processing over a cellular structure in which decisions for each cell are made purely locally but subject to contents of the neighboring cells. It has been shown that such iterative computations converge, in most cases, to solutions that correspond to a local minimum of some optimization criterion. In probabilistic relaxation, also called continuous relaxation, we seek to maximize the probability of a cell occupying a value subject to the value being compatible with those in the neighboring cells. On the other hand, in discrete relaxation, at each iteration we adjust the value of a cell in order to satisfy certain discrete constraints on the value with regard to the values in the neighboring cells. The main attraction for relaxation-based procedures lies in the fact that they are well suited to parallel implementations, especially on cellular array processing architectures, in light of the inherently local computations involved, meaning that to update a value in a cell the system need communicate with only the neighboring cells.

Since fundamental to a graph is the connection of a node with its neighboring nodes, relaxation-based processing extends very naturally to computations over graphs, as has been demonstrated by Haralick and Shapiro [22], [40], Haralick and Elliot [20], and Kitchen and Rosenfeld [30]. In these extensions to graph-theoretic computations, we first assign to each node in a graph, which could correspond to our scene object graph, all possible labels corresponding to the model features on the basis of some similarity criteria. These label sets are then pruned by enforcing the relational constraints, as observed in the scene, between different pairs of nodes in the scene graph. If the iterative application of this constraint enforcement leads to a unique label for each node in the scene graph, we have accomplished scene interpretation via discrete relaxation.

We will now demonstrate how we have set up the computations in the pruning module. For the candidate model object under consideration, the input to the pruning module consists of the model graph, the scene graph, and the assignment-consistency table produced by the filtering module.

The pruning module first extracts from both the scene and the model graphs the subgraphs centered at each of the nodes and composed of the immediately neighboring nodes; these subgraphs will be called *elementary trees* (ET's). For illustration, for the scene and the model graphs shown in Fig. 7(c) and (d), the corresponding collections of ET's are displayed in Fig. 11(a) and (b), respectively. So, if the model graph has $N$ surfaces and, consequently, $N$ nodes, there will be $N$ ET's of the type indicated.

There are two important things to note about decomposing the model and scene graphs into ET's. One, each ET captures the local relations that exist at each node of either the model graph or the scene object graph. And, two, as will be shown by



(a) (b) (c) (d)

(e) (f)

Fig. 11. For the scene and the model graphs shown in Fig. 7(c) and (d), here in (a) and (b), respectively, are the collections of the elementary trees. Shown on the left in (d) is the ET rooted at node 1 in the scene graph, and on the right the ET rooted at node $d$ in the model graph. The dashed lines here correspond to the node consistencies as given by the table in (c). Also, the + and – indicate convex and concave edges, respectively. When the pruning module is invoked with the assignment-consistency table shown in (c) here, then after only one pass through discrete relaxation the output assignment-consistency table is as shown in (e). The assignment-consistency table in (f) is obtained by enforcing the constraint that a model label not be assigned to more than one scene node.

us presently, subgraph isomorphism can be established between two ET's in polynomial time.[7]

We will exploit the polynomial time computability of subgraph isomorphism between ET's in a computational scheme which causes significant pruning of the search space that remains after the processing performed by the filtering module. Basic to this computational scheme is the establishment of local compatibilities, defined in Section IV, between the nodes of the scene graph and those of the candidate model graphs. The reader should note that a node from the scene graph is locally compatible with a node from the model graph if and only if there exists a subgraph isomorphism between the ET's rooted at the two nodes. We will now discuss the flow of control in this computational scheme.

The pruning module scans the assignment-consistency table, element by element, and when it runs into a "1", it checks for local compatibility between the scene and the model nodes involved. For example, suppose the assignment-consistency table is as shown in Fig. 11(c). The pruning module will now examine each element of this table, from left to right and top to bottom. The first entry of 1 it sees in the table corresponds to node 1 from the scene and to node $a$ from the model. The entry of 1 at this location in the table will be retained only if the pruning module is able to establish local compatibility between the scene node 1 and the model node $a$; establishment of this local compatibility corresponds to subgraph isomorphism between the ET rooted at scene node 1 and the ET rooted at the model node $a$.

A polynomial time implementation of testing for local compatibility can be carried out by applying, after a slight modification, the bipartite matching algorithm described in Appendix A to the two ET's involved and checking for the existence of a complete match. The modification consists of first augmenting the attribute list at each of leaf nodes in the ET's by the attributes of the arcs connecting the leaf nodes with the root node, and then applying the bipartite matching algorithm.

[7] As intuition would suggest, establishing subgraph isomorphisms between the scene ET's and the model ET's does not imply the existence of a subgraph isomorphism between the scene graph and the model graph, this being the case even when we have a distinct model ET for each scene ET.

For example, to check whether or not there exists local compatibility between, say, the scene node 1 and the model node $d$, we apply the bipartite matching algorithm to the node sets given by $\Gamma(1)$ and $\Gamma(d)$, where, as the reader will infer from its earlier definition, $\Gamma(x)$ designates the list of leaf nodes in the ET rooted at node $x$. The ET's rooted at 1 in the scene and $d$ in the model are shown in Fig. 1(d). We therefore apply the bipartite matching algorithm to the node sets $\{3,4\}$ and $\{c,b\}$ after the attribute "convex" has been included in the list of attributes for node 3 and so on. We maintain that if, after such augmentation of node attributes there exists a complete match, as discovered by the bipartite matching algorithm, between $\Gamma(1)$ and $\Gamma(d)$, then there must exist a subgraph isomorphism between the ET's rooted at the two nodes 1 and $d$. Of course, for the example being discussed, there does not exist such a complete match, owing to the "absorption" of arc attributes by the nodes; therefore, scene node 1 is locally incompatible with the model node $d$. Hence, the pruning module will remove the corresponding "1" from the assignment-consistency table. It is important to note that for subsequent scans through the assignment-consistency table, the scene node 1 will not be considered to be in node consistency with the model node $d$.

For the scene and the model objects shown in Fig. 7(a) and (b), after the first pass through the assignment-consistency table by the pruning module the table looks as shown in Fig. 11(e). In our current implementation, during each pass through the table, the pruning module is only aware of the table entries as they existed at the end of the previous pass. Note that the table is used by the bipartite matching algorithm to figure out the matching between the nodes. The bipartite matching algorithm will assume that a scene node matches a model node provided there is a "1" in the corresponding position in the assignment-consistency table and provided the "absorbed" arc attributes are identical.

Each pass through the assignment-consistency table constitutes one iteration of discrete relaxation. For each scene node, corresponding to a row of the table, the set of column headings for the "1" entries constitutes the set of allowable labels from the model. For example, in Fig. 11 for the scene node 1, the set of labels before the invocation of the pruning module is $\{a, c, d, e\}$.

Enforcement of local relational constraints by the application of a single pass of the pruning module changes this set of permissible labels to $\{c, e\}$.

The iterative application of discrete relaxation is stopped when no further entries from the assignment-consistency table are deleted. For simple objects, the number of iterations required is very small. For example, when the pruning module is invoked with the assignment-consistency table shown in Fig. 11(c) as input, only one iteration of discrete relaxation is required to converge to the final form of assignment-consistency table, which is shown in Fig. 11(e).

The reader should note that if at any time during the process of discrete relaxation all the entries in any row of the assignment-consistency table become zero, the candidate model is rejected. This condition is tantamount to there being no model features available for matching with an observed scene feature.

In the final assignment-consistency table produced by the iterative application of discrete relaxation, we now insist that a model label not be assigned to more than one scene node, meaning that no column of the table should contain more than one entry of "1". This constraint is easily enforced by visiting each row of the table, locating rows containing a single entry of "1", and then eliminating any other "1"s in the same column. For example, in scanning the table shown in Fig. 11(e), the first row that contains a single entry of "1" corresponds to scene node $c$. We now insist that the model label $c$ be reserved for the scene node 3 and, therefore, delete the two other "1"s in the third column. When we scan through the entire table in this fashion, the final table output by the pruning module is as shown in Fig. 11(f).

### D. The Ambiguity Resolution Module

As one would expect, although the pruning module eliminates large chunks of the search space, ambiguities still remain, both with regard to the identity of the object and its pose. For each candidate model object that survives all the processing until the end of the pruning module, pose ambiguities manifest themselves in the form of there being more than one entry of "1" in a row of the table. Pose ambiguities are resolved in the following manner.[8]

We first estimate the pose of the object by using all those scene features that have been matched uniquely with model features; such matches correspond to single entries of "1" in the rows of the table. The mathematics for such a pose estimation is described in Appendix B and is based on the principles advanced by Faugeras and Hebert [12]. To use this mathematics, we first construct pairs of extrinsic attributes, such as outward normals for planar surfaces, axis directions for cylindrical surfaces, centroids of areas, etc., for the corresponding uniquely matched features from the scene and the model. We then invoke the equations presented in Appendix B for optimum estimation of the rotational and translational components of the pose transform associated with the scene object.

For example, for the scene object shown in Fig. 9, the assignment-consistency table, after the pruning module is done with it, is shown in Fig. 12. The pose estimation algorithm recognizes that the scene features $\{1, 2, 4, 5, 7, 8\}$ are uniquely matched with the model features $\{a, b, c, d, e, f\}$ as labeled in model 11 in Fig. 6. The algorithm now pairs up the extrinsic attributes of 1 and $a$, of 2 and $b$, etc., and computes from these pairs of attribute values the rotational and the translational components of the pose.

[8] A most important point to note is that even when the pruning module ends up matching a unique model feature with every scene feature, the model object may still be untenable on account of geometrical inconsistencies. We will address this point later in the paper.

| | a | b | c | d | e | f | g | h | i | j | k |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | . | . | . | . | . | . | . | . | . | . |
| 2 | . | 1 | . | . | . | . | . | . | . | . | . |
| 3 | . | . | . | . | . | . | . | 1 | . | . | 1 |
| 4 | . | . | 1 | . | . | . | . | . | . | . | . |
| 5 | . | . | . | 1 | . | . | . | . | . | . | . |
| 7 | . | . | . | . | 1 | . | . | . | . | . | . |
| 8 | . | . | . | . | . | 1 | . | . | . | . | . |
| 9 | . | . | . | . | . | . | 1 | 1 | . | . | 1 |

Fig. 12. This assignment-consistency table is produced by the pruning module for the scene graph shown in Fig. 9(d). This table corresponds to model #1 shown in Fig. 6. For the scene object shown, all other models would be rejected by the filtering module.

*Pose Estimation and Validation:* As discussed in Appendix B, the pose estimation and validation module needs at least two unique assignments of model labels to object nodes for pose estimation. Also, when only two features are available for pose estimation, the module makes sure that their orientation vectors, such as surface normals or cylinder axes, are not parallel. When more than two features are available, this check is not carried out explicitly, because in this case the validity checks described later in this subsection would report failure if there is not adequate independent information contained in the pairwise associations of scene and model orientation vectors.

This module reports failure if the number of scene nodes that possess unique assignments in the assignment-consistency table is less than two, or when the number is two but the orientation vectors are nearly parallel, or when one of the validation criteria described below is violated.

The mathematics presented in Appendix B and in [6] and [12] for pose estimation computes an optimum pose transform, optimum in the sense that the computed pose minimizes the mean-squared error between the extrinsic attributes of the scene object and those of the model object. Since computations that seek optimum values involve some kind of averaging over the available data, it is possible that the resulting optimum pose transform may not be valid. The computed pose transform may also not be valid if the information supplied is not sufficiently independent. For example, if we seek to compute the pose of an object from, say, three separate but almost coplanar surfaces, using their surface normals for the computation of the rotation matrix, the information supplied to the algorithm for the computation of the pose transform may not be sufficiently linearly independent, leading to an unreliable pose transform matrix. The validity of the computed pose is checked by applying the pose transform to each of the extrinsic attributes for each feature that contributed to the computation of the transform; in assessing the results, we treat different types of features and different types of attributes separately. We make distinctions between the following types.

1) We compare the surface normals of the corresponding planar surfaces in the scene and the model by computing the following metric

$$d_{normals} = \frac{1}{2}(1 - (\boldsymbol{R}\boldsymbol{v}_s) \cdot \boldsymbol{v}_m)$$

where $\boldsymbol{R}$ is the rotational component of the computed pose and $\boldsymbol{v}_s$ and $\boldsymbol{v}_m$ are, respectively, the normals associated with

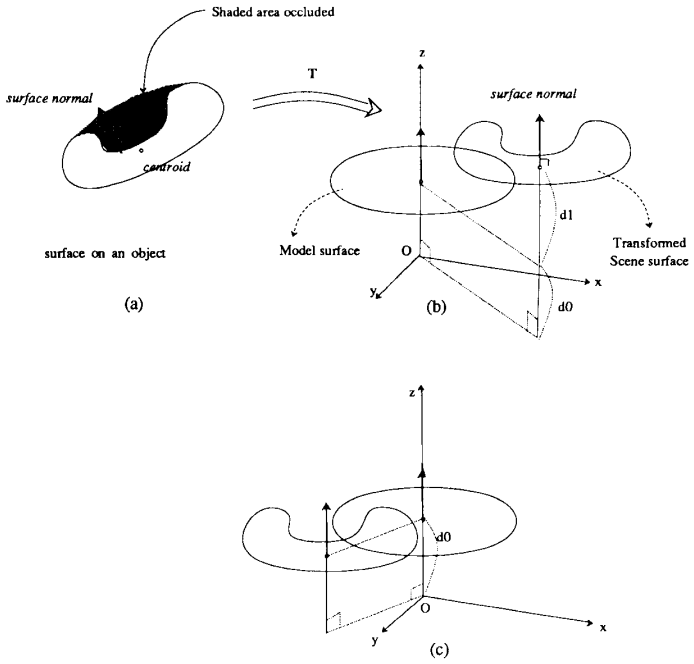



Fig. 13. (a) An object surface that is not completely visible due to occlusion. (b) The metric $d_{p\_centroid\_perp}$ computes the perpendicular distance between the model surface and the scene surface as transformed into the model coordinate frame. (c) As shown here, even if the perpendicular displacement $d_1$ is zero, the transformed scene surface still has three degrees of freedom with respect to the model surface. To constrain these three remaining degrees of freedom, we use the metric $d_{p\_centroid\_total}$.

corresponding scene and the model planar surfaces. The metric $d_{normals}$ equal 0 when $\boldsymbol{R}$ is free of errors. On the other hand, it equals 1 for a scene surface whose normal after transformation is exactly opposite to the model normal.

2) The centroids[9] of the corresponding planar surfaces in the scene and the model are compared by using the following metric

$$d_{p\_centroid\_perp} = |\boldsymbol{T}\boldsymbol{p}_s \cdot \boldsymbol{R}\boldsymbol{v}_s - \boldsymbol{p}_m \cdot \boldsymbol{v}_m|$$

where $\boldsymbol{T}$ is the computed pose transform, of which $\boldsymbol{R}$ is the rotational component, and $\boldsymbol{p}_s$ and $\boldsymbol{p}_m$ are the centroids of the scene and the model surfaces, respectively. The subscript $p\_centroid\_perp$ signifies the use of this metric for planar surfaces only. The geometric significance of this metric is illustrated in Fig. 13, where, in Fig. 13(a), we have shown schematically a scene surface which is only partially visible due to some occlusion. The important point to note here is that for a planar surface any occlusion will affect only the location of the centroid in the plane of the surface and not the perpendicular to the surface. In other words, the equation of the scene surface, which is characterized by the surface normal and the perpendicular distance of the surface from the origin, is not altered by occlusion. The metric shown above gives us the distance $d1$ illustrated in Fig. 13(b). If this metric exceeds a threshold, the pose transform $\boldsymbol{T}$ is rejected. It is of course true that as illustrated in Fig. 13(c) the metric shown above will constrain the pose transform with regard to only one degree of freedom—the direction normal to the model surface; the other degrees are constrained by the application of the following metric, which measures the total distance between the model surface and the scene surface centroids,

$$d_{p\_centroid\_total} = |\boldsymbol{T}\boldsymbol{p}_s - \boldsymbol{p}_m|.$$

Note that the metric $d_{p\_centroid\_perp}$ must be applied for the acceptance or rejection of the pose transform $\boldsymbol{T}$ *before* the metric $d_{p\_centroid\_total}$ is applied for the same purpose because of the occlusion dependency of the latter and because, even in the absence of occlusion, the latter metric is incapable of providing a constraint on just that displacement between the surfaces which is perpendicular to the surfaces. Despite its dependency on occlusion, we have found it necessary to use the $d_{p\_centroid\_total}$ metric during verifications of pose transforms of objects that show some symmetry. To reduce its occlusion dependency, we use a variable threshold for the metric, as given by

$$thresh\_dist' = thresh\_dist \frac{area_m}{area_s}$$

where $thresh\_dist$ is a user supplied threshold and where $area_s$ and $area_m$ are the areas of the scene and the model

[9] Actually, any pair of points, one on the model surface and the other on the corresponding scene surface, would lead to the same value for the $d_{p\_centroid\_perp}$ metric, as long as the transformed version of the scene surface is parallel to the model surface. (It is this freedom with regard to the choice of the locations that makes the metric insensitive to occlusion.) We use the centroids simply to ensure that the points are not too close to the boundaries of the surfaces where the computations of the surface attributes are more prone to error.

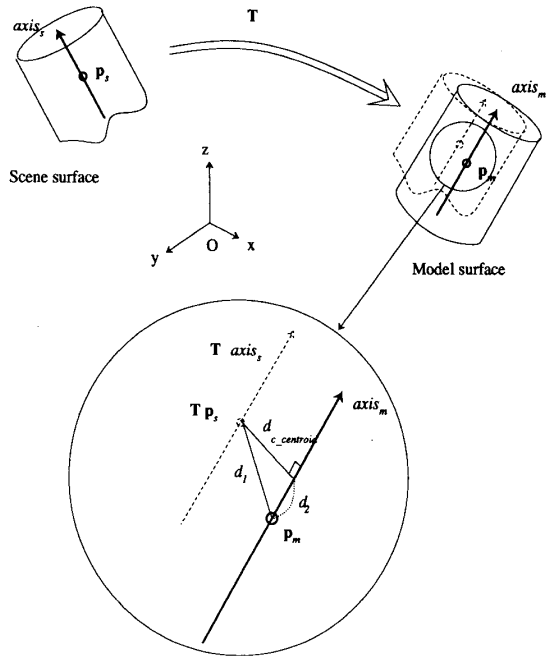



Fig. 14. The cylindrical surface at upper left is as found in a scene. This surface is transformed by the pose-transformation matrix to yield the corresponding cylindrical surface in model coordinates, as shown by the dashed lines at the upper right. The exploded view at the bottom illustrates the distances $d_1$, $d_2$, and $d_{c\_centroid}$.

surfaces, respectively. If this threshold is exceeded by $d_{p\_centroid\_total}$, the pose transform is rejected.

3) We compare the corresponding cylindrical surfaces by computing the $d_{axis}$ metric, which gives us a measure of angular error between the direction of the axis of the cylindrical surface in the scene and direction of the axis of the corresponding cylindrical surface in the model, after the scene has undergone the rotational transformation $\boldsymbol{R}$. The metric is given by

$$d_{axis} = (1 - |(\boldsymbol{R}axis_s) \cdot axis_m|)$$

where $axis_s$ and $axis_m$ are, respectively, the unit vectors associated with the corresponding scene and the model cylindrical axes. The metric $d_{axis}$ equals 0 when the axis of the scene cylindrical surface is exactly aligned with that of the corresponding model surface, or exactly opposite to it. On the other hand, it becomes 1 when the axis of the scene cylindrical surface, after transformation, is orthogonal to the corresponding model surface axis. When $d_{axis}$ exceeds some preset threshold, the candidate model object is rejected.

4) The centroids[10] of the corresponding cylindrical surfaces in the scene and the model are compared by evaluating the following metric:

$$d_{c\_centroid} = |d_1 - d_2|$$

where

$$d_1 = \boldsymbol{T}\boldsymbol{p}_s - \boldsymbol{p}_m$$
$$d_2 = (d_1 \cdot axis_m)axis_m$$

where $axis_m$ stands for a unit vector along the axis of the model cylinder. As shown in Fig. 14, $|d_1|$ corresponds to the Euclidean distance between the scene centroid as transformed into model coordinates and the model centroid, and $d_2$ to the projection of $d_1$ on to the axis of the model cylinder. The metric $d_{c\_centroid}$ measures the orthogonal distance from the transformed centroid to the model axis. Clearly, a candidate model cylinder should be rejected if $d_{c\_centroid}$ exceeds a certain threshold.

5) For spherical surfaces, we compute the distance between the centroid of the model sphere and the scene sphere transformed by $\boldsymbol{T}$.

6) For conical surfaces, the metric for comparing the directions of the axis is the same as was used for comparing the surface normals in the planar case. Note that we could not have used the $d_{axis}$ metric developed for the cylindrical case because that metric was designed to accommodate 180° ambiguity in the direction of one axis with that of the other. In that sense, the conical case is more similar to the planar case, because there are no such ambiguities to deal with.

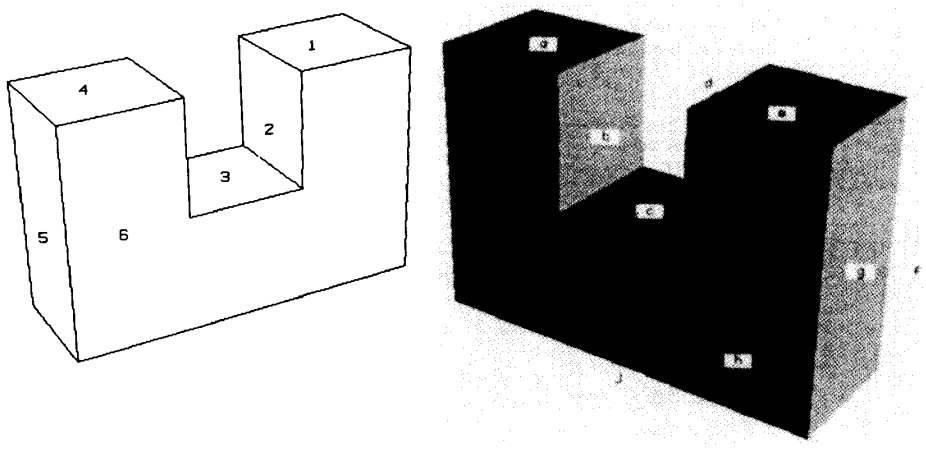
7) While the metric in item 6 measures how the two conical surfaces line up with regard to orientation of their axes, the following metric measures the displacement that is approximately along the direction of alignment (it is exactly so if the model axis is collinear with the transformed scene axis). The metric is given by

$$d_{c\_centroid} = |\boldsymbol{T}apex_s \cdot \boldsymbol{R}axis_s - apex_m \cdot axis_m|$$

where $axis_m$ and $axis_s$ are the axes, respectively, of the model cone and the scene cone, and $apex_m$ and $apex_s$ are, respectively, the apex of the model cone and the apex of the scene cone. In general, of course, the apex of the scene cone may not be visible. However, its position can be estimated using simple trigonometry if we can estimate from the visible surface the slant angle of the cone and the radius of the circle corresponding to at least one visible point.

After a pose transform is verified, the next task is to disambiguate those rows of the assignment-consistency table that contain more than one entry of "1". Disambiguation is carried out by applying the pose transform to the extrinsic attributes of the scene nodes that correspond to these rows, and comparing the results obtained with the extrinsic attributes of the candidate model nodes. For example, for the assignment-consistency table output by the pruning module and shown in Fig. 12—this table corresponds to the scene object shown in Fig. 9and the model object #11 in Fig. 6—a pose transform was computed from those entries of the table that are characterized by the occurrence of a single "1" in the rows. This pose transform was not rejected during verification when we checked it separately against each of the features that contributed to the estimation of the transform. When the estimated pose transform was applied to the extrinsic attributes of scene feature 9, the extrinsic attributes being surface

[10]For lack of a better term, we are using the word *centroid* to designate a point on the axis of a cylindrical surface, even when the surface is only partially visible. Centroid will stand for the middle point of the axis corresponding to the visible portion of the cylinder. However, as was the case with planar surfaces, any two points, one on the axis of the scene cylinder and the other on the axis of the corresponding model cylinder, would do, as long as the two axes are parallel after the scene axis is transformed into the model space.




(a) (b)

| | a | b | c | d | e | f | g | h | i | j |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | . | . | 1 | . | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 5 | . | . | 1 | . | . | 1 | 1 | 1 | 1 | 1 |
| 6 | . | . | . | . | . | 1 | . | 1 | . | . |

(c)

| | a | b | c | d | e | f | g | h | i | j |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | . | . | . | 1 | . | . | . | 1 | . |
| 2 | . | 1 | . | 1 | . | . | . | . | . | . |
| 3 | . | . | 1 | . | . | . | . | . | . | . |
| 4 | 1 | 1 | . | 1 | 1 | . | 1 | . | 1 | 1 |
| 5 | . | . | . | . | 1 | . | 1 | . | 1 | 1 |
| 6 | . | . | . | . | . | 1 | . | 1 | . | . |

(d)

| | a | b | c | d | e | f | g | h | i | j |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | . | . | . | . | . | . | . | 1 | . |
| 2 | . | 1 | . | . | . | . | . | . | . | . |
| 3 | . | . | 1 | . | . | . | . | . | . | . |
| 4 | 1 | . | . | 1 | 1 | . | 1 | . | 1 | 1 |
| 5 | . | . | . | . | 1 | . | 1 | . | 1 | 1 |
| 6 | . | . | . | . | . | 1 | . | 1 | . | . |

(e)

Fig. 15. (a) A scene object in a random pose. (b) The corresponding model object with surface labels as shown. Many surfaces on this object have identical intrinsic attributes; for example, surfaces $a, b, c, d$ and $e$ have identical areas. (c) The assignment-consistency table for the scene object of (a) and the model of (b). (d) The assignment-consistency table as output by the Pruning Module. (e) The assignment-consistency table resulting from assigning model label $b$ to scene surface 2 and applying the processing steps in Fig. 16.

normal and centroid, we obtained best results with the extrinsic attribute values of model node $g$. Therefore, the competing model labels $h$ and $k$ were rejected for scene node 9.

*Hypothesis Disambiguation and Pose Estimation for Objects of High Symmetry:* As the reader will recall, the filtering and the pruning modules compare surface features on the basis of only their intrinsic attributes, such as surface types, areas, number of neighbors, etc. In addition to utilizing these criteria, the pruning module also enforces certain relational constraints, but here again the comparisons between the scene object and a candidate model object do not utilize extrinsic attributes which are dependent upon the pose of the scene object. As a result, if the scene and the model objects possess many similar surfaces, a common occurrence with objects of high symmetry, the assignment-consistency table produced by the pruning module will possess very few rows with single entries of "1" in them.

For example, for the scene and the model objects in Fig. 15(a) and (b), Fig. 15(c) and (d) shows, respectively, the assignment-consistency tables *before* and *after* the invocation of the pruning module. As is clear from Fig. 15(d), only one scene feature has a unique model label.

At first sight, the reader might fault our filtering and pruning strategies for not yielding a larger number of unique assignments for scene features than what is shown in Fig. 15(d). However, it is important to note that the difficulty is caused not by any serious shortcomings in the filtering and the pruning modules, but by the fact that a pose is not uniquely defined for objects of high symmetry, meaning that for such objects there can exist many one–one assignments of scene features to model features, each yielding a different but valid pose transformation. We will now explain our procedure for extracting one such valid pose. This we will do with the help of the scene and the model objects shown in Fig. 15. The flow of control for this procedure is shown in Fig. 16.

To estimate a pose from assignment-consistency tables containing only a few unique model-to-scene assignments, we first choose a scene node for which we have the least number of permissible model labels under the proviso that this least number exceeds unity. For the example in Fig. 15(d), scene node 2 will be chosen first. (It will serve no purpose to select scene node 3, which already has a single model label, since the subsequent processing by the pruning module in Fig. 16 will leave the entire table unchanged. Selecting the row corresponding to scene node 2 and making all entries "0" except one introduces a perturbation into the table that forces the pruning module to also prune out some of the excess labels for the other scene nodes.) We now assign to this node the first available model label, which is $b$; we say "available" because if the currently selected label does not work out from the standpoint of leading to a viable pose transform, the system will select the next label. In other words, if necessary the system must backtrack over the set of model nodes in the second row.[11]

[11] It is important to appreciate the fact that even in the worst case the backtracking does not have to be carried out over all possible model label choices for the scene nodes. Backtracking is limited to a small subset of scene nodes, only those that are needed for the computation of a pose transform. For

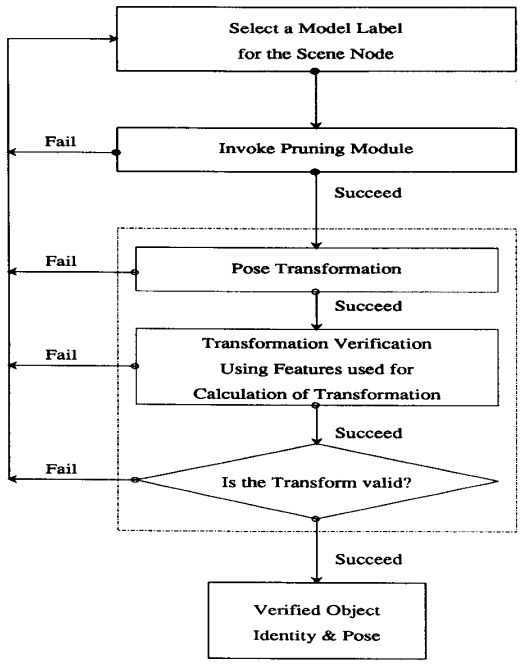



Fig. 16. The processing steps and the flow of control which replace the ambiguity resolution module in Fig. 8 for objects of high symmetry.

After we have assigned a model label to scene node 2, then in accordance with the flow of control in Fig. 16 the pruning module is invoked again so that only the compatible assignments are retained for the other scene nodes. This action of the pruning module is identical to that described before. The assignment-consistency table as produced by this invocation of the pruning module is as shown in Fig. 15(e).

For highly symmetric objects, the assignment-consistency table, as produced by the pruning module (second block from the top in Fig. 16), will still contain only a few unique model label assignments for the scene features. In principle, one could repeat the process of first assigning to scene nodes one of the permissible model labels and then invoking the pruning module, until one obtained a one–one mapping from scene to model. However, our approach is different and, we believe, computationally more efficient. We send the assignment-consistency table to the module for pose estimation. As was mentioned before, the pose estimation and validation module will report failure if it does not get at least two scene nodes with nonparallel orientation vectors, or if one of the validation criteria discussed in the preceding subsection is not satisfied.

## VI. Experimental Results

We have tested our system on many scenes composed of single and multiple objects. The model library for these experiments consisted of the 12 objects shown in Fig. 6. The PADL system [38] was used for the modeling of each object. PADL is a constructive solid geometry (CSG) based modeling system; an object is represented by a composition tree whose nonterminal nodes are regularized versions of the set operators, union, difference, and intersection, and whose terminal nodes are the primitives of the system, namely block, cylinder, cone, etc. The terminal nodes also contain information on the placement and scaling of the primitives, such information at each terminal node being in the form of a $4 \times 4$ homogeneous transformation matrix. Jeffrey Lewis at the Robot Vision Lab has modified the PADL software to yield via an interactive mode the graph representations, discussed in Section IV, for objects. It is interesting to note that for constructing attribute frames, most of the attributes do not have to be computed by the modeling software as they are either user-specified or trivially computed from the information supplied by the user. For example, the attributes *axis* and *radius* for cylindrical surfaces must be supplied by the user at the time a model containing a cylindrical surface is created, and the attribute *normal* is easily computed from the user-supplied transformations for the placement of the primitives in the composition tree. On the other hand, there are three attributes, *area, adjacency,* and *edge_type,* whose computation in the PADL framework requires more effort. Some of the modifications carried out by Jeffrey Lewis allow automatic computation of such attributes.

We will now report results on two of the many experiments we have conducted successfully in our lab. These experiments involved both single-object scenes and multiobject scenes with varying degrees of occlusion. For the discussion to follow, we have chosen one single-object scene and one multiobject scene.

### A. Single-Object Scene Experiments

A single-object scene is shown in Fig. 17(a), its structured-light scan in Fig. 17(b) and the corresponding preprocessed map in Fig. 17(c).[12,13] Since the preprocessor outputs the attribute frames for each of the segmented surfaces in Fig. 17(c), it is a simple matter to construct from them a graph representation of the type discussed in Section IV. This graph is then analyzed by the method discussed in the previous sections.

For the scene of Fig. 17, all the models shown in Fig. 6 were rejected except for model 10. Models 1–8, 11, and 12 were rejected by the filtering module and model 9 by the pruning module. For the model that eventually survives, model 10, the assignment-consistency tables at various points in the control flow diagram of Fig. 8 are shown in Fig. 18. The assignment-consistency table input to the pruning module is shown in Fig. 18(a), the assignment-consistency table after one iteration of discrete relaxation in the pruning module is shown in Fig. 18(b) and after two iterations in Fig. 18(c); no change from Fig. 18(b) to (c) indicates convergence in only one iteration. In the assignment-consistency table of Fig. 18(c), six of the scene nodes have unique model labels. Scene node 6 has two labels for very understandable reasons; the labels $d$ and $f$ (see model 10 of Fig. 5) correspond to the two planar parallel faces shown in Fig. 17(c) and their intrinsic attribute values are identical.

the example of Fig. 15, if no pair of model labels for scene nodes 2 and 3 lead to a valid pose transform, then we could not possibly construct a complete matching between the scene nodes and the model nodes, and therefore the model would have to be rejected.

[12]The light stripe image shown in Fig. 17(b) is from the camera viewpoint and the preprocessed range map corresponds to the projector viewpoint. In processing data from structured light scanners, it has become conventional to show results from the stripe projector viewpoints, which is what we have done here. Of course, it does not make any sense to show the stripe images from the projector viewpoint because the objects would not be discernible in such images.

[13]The preprocessor is discussed in [6]. The preprocessor first computes surface normals from the range map and then uses a combination of range and surface normal discontinuities in a region growing framework to segment the range map into a set of analytically continuous surfaces. The preprocessor also computes the values of the various attributes defined for the different surfaces.

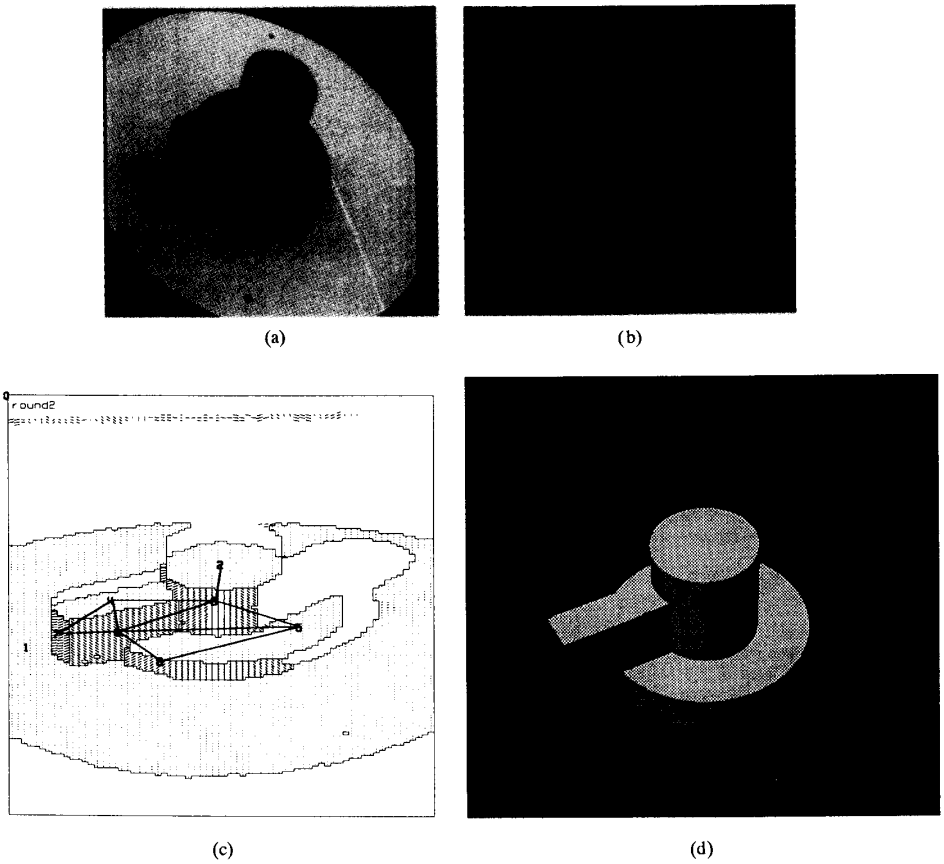



Fig. 17. (a) A scene consisting of a single object in a random position and orientation in the work area of the robot. (b) A light-stripe image of the scene made with a robot hand-held structured-light scanner. (c) The preprocessed range map showing segmented surfaces and the resulting graph representation of the scene. (d) To domonstrate successfuly object identification and successful calculation of pose, this figure was generated by invoking the PADL solid modeling system with the computed object identity and pose transofrm.

Since unique model labels are available for more than one scene feature, the pose of the scene object is estimated and, in this case, successfully validated. The pose transform computed from the uniquely matched scene and model surfaces is shown in Fig. 18(d). In accordance with the discussion in Appendix B, the computed axis of rotation $\boldsymbol{n}$ and the angle of rotation $\theta$, both shown in Fig. 18(d), correspond to the two parameters, one vector and one scalar, in the quaternion representation of a rigid body rotation.

The ambiguity resolution module then takes over and resolves the ambiguities present in the model feature assignment for scene feature 6. The label $d$ is rejected because, on the basis of the pose transform, its surface normal is exactly opposite to the transformed surface normal for scene surface 6; this fact is discovered by the value of the metric $d_{normals}$ exceeding its allowed threshold. The final assignments of model labels to scene nodes are shown in Fig. 18(e). In Fig. 17(d) we have shown the PADL generated model object after it is transformed by the pose matrix in Fig. 18(d); the similarity of the pose with that of the object in the scene is evident.

To give the reader some idea of the efficiency of the computations involved, on a SUN3 workstation it took only 0.1 s of CPU time to compute the identity and pose of the object in Fig. 17 from the preprocessed map in Fig. 17.

### *B. Multiobject Scene Experiments*

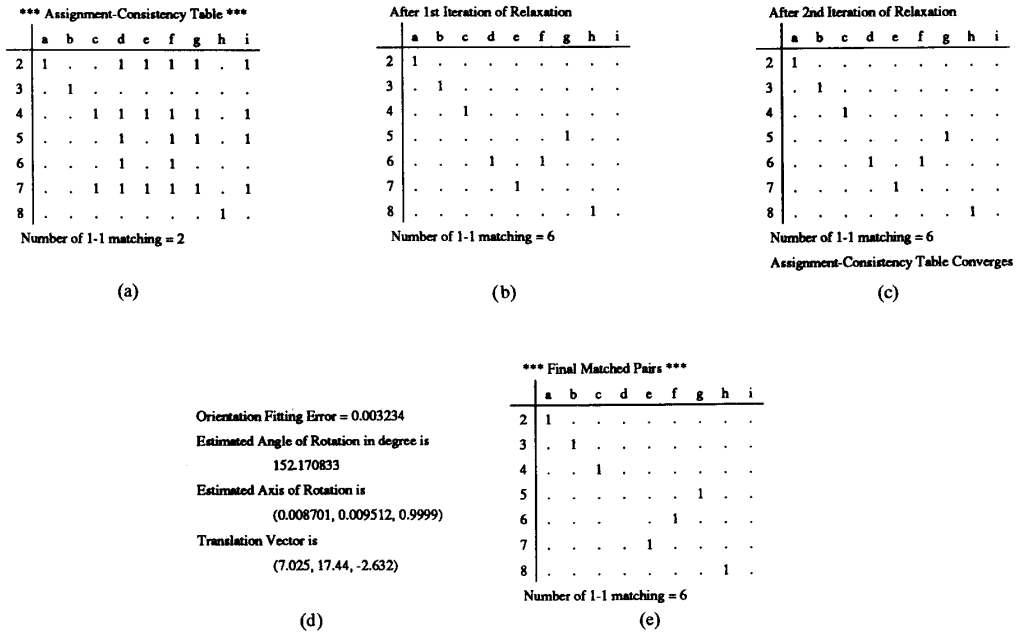
Since ultimately any strategy for object recognition and pose estimation must work in the presence of occlusions, we have investigated the application of our procedure to multiobject scenes like those shown in Figs. 1 and 19. The latter scene will be used for the discussion here.

As with single-object scenes, the range map of a multiobject scene is first processed to yield surfaces and the edges between them. Range discontinuity edges are then used to segment the entire map into separate scene graphs, each possibly corresponding to a model object. For the scenes of Figs. 1(a) and 19(a), the various scene graphs constructed are shown, respectively, in Figs. 1(c) and 19(c). The following labels will designate the different scene graphs in Fig. 19(c):

$$G_1 = \{1, 2, 9, 13, 15, 16, 25\}$$
$$G_2 = \{22, 24, 29, 31, 32, 34\}$$
$$G_3 = \{26, 28, 30, 33, 38\}.$$

For brevity, we will not list the surfaces in the other two graphs that are shown in Fig. 19(c) because the system is not able to find any model objects corresponding to them.

In general, due to the presence of a larger number of occlusions, the quality of a range map for a multiobject scene will be



*** Assignment-Consistency Table ***

| | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 1 | . | . | 1 | 1 | 1 | 1 | . | 1 |
| 3 | . | 1 | . | . | . | . | . | . | . |
| 4 | . | . | 1 | 1 | 1 | 1 | 1 | . | 1 |
| 5 | . | . | . | 1 | . | 1 | 1 | . | 1 |
| 6 | . | . | . | 1 | . | 1 | . | . | . |
| 7 | . | . | 1 | 1 | 1 | 1 | 1 | . | 1 |
| 8 | . | . | . | . | . | . | . | 1 | . |

Number of 1-1 matching = 2

(a)

After 1st Iteration of Relaxation

| | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 1 | . | . | . | . | . | . | . | . |
| 3 | . | 1 | . | . | . | . | . | . | . |
| 4 | . | . | 1 | . | . | . | . | . | . |
| 5 | . | . | . | . | . | . | 1 | . | . |
| 6 | . | . | . | 1 | . | 1 | . | . | . |
| 7 | . | . | . | . | 1 | . | . | . | . |
| 8 | . | . | . | . | . | . | . | 1 | . |

Number of 1-1 matching = 6

(b)

After 2nd Iteration of Relaxation

| | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 1 | . | . | . | . | . | . | . | . |
| 3 | . | 1 | . | . | . | . | . | . | . |
| 4 | . | . | 1 | . | . | . | . | . | . |
| 5 | . | . | . | . | . | . | 1 | . | . |
| 6 | . | . | . | 1 | . | 1 | . | . | . |
| 7 | . | . | . | . | 1 | . | . | . | . |
| 8 | . | . | . | . | . | . | . | 1 | . |

Number of 1-1 matching = 6

Assignment-Consistency Table Converges

(c)

Orientation Fitting Error = 0.003234

Estimated Angle of Rotation in degree is

152.170833

Estimated Axis of Rotation is

(0.008701, 0.009512, 0.9999)

Translation Vector is

(7.025, 17.44, -2.632)

(d)

*** Final Matched Pairs ***

| | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 1 | . | . | . | . | . | . | . | . |
| 3 | . | 1 | . | . | . | . | . | . | . |
| 4 | . | . | 1 | . | . | . | . | . | . |
| 5 | . | . | . | . | . | . | 1 | . | . |
| 6 | . | . | . | . | . | 1 | . | . | . |
| 7 | . | . | . | . | 1 | . | . | . | . |
| 8 | . | . | . | . | . | . | . | 1 | . |

Number of 1-1 matching = 6

(e)

Fig. 18. (a) Input to the pruning module consists of this assignment-consistency table for the scene of Fig. 17(a). (b) The assignment-consistency table after one iteration of discrete relaxation in the pruning module. (c) The assignment-consistency table after two iterations of discrete relaxation. (d) The computed pose transform. (e) The final assignments of model labels to scene surfaces.

inferior to that for a single-object scene. In order to cope with such occlusions, we must use inequalities in the feature matching criteria discussed in Section IV. Such inequalities, although necessary, lead to the possibility that an occluded feature will in general match many model features. For example, surface 25 has a diminished value of *area* attribute due to occlusion, therefore, this scene surface matches every model surfaces except *e* on model 11 (see Fig. 6 for model surface labels). This fact is illustrated by the last row of the assignment-consistency table produced by Filter 3 for scene object $G_1$, as shown in Fig. 20(a). Similarly, scene surface 38 has a diminished value of the area attribute due to the poor quality of the range data in that part of the scene. Scene surface 38 also suffers from the fact that it is adjacent to only one other scene surface. If the reader will recall the definition of node consistency, both these shortcomings cause many model surfaces to be declared matchable with scene surface 38, as illustrated by the last row of the assignment-consistency table produced by Filter 3 for the scene graph $G_3$, as shown in Fig. 22(a). As shown there, all the surfaces except *e* on model 11 are declared matchable with scene surface 38.

Since the pruning module uses relational information and since occlusions in multiobject scenes may prevent the discovery of some of the relations between surfaces, the performance of this module may also suffer. For example, when the pruning module is invoked for the scene graph $G_3$, the input assignment-consistency table is as shown in Fig. 22(a) and the output as shown in Fig. 22(b). As is clear from the output table, only two scene surfaces, 30 and 33, get unique model surface assignments after the pruning module has done its job. Compare this with the results obtained for the scene graph $G_1$. Its input and output assignment-consistency tables for the pruning module are shown in Fig. 20(a) and (b). In this case, through the action of the pruning module, five of the seven scene surfaces acquire unique model surface labels.

It is interesting to note that the surfaces that do acquire unique model surface labels for scene graph $G_2$ are not capable of generating a unique pose transform for the object and the alternative implementation of the ambiguity resolution module presented in Section V-D must be invoked. As shown in the tables in Fig. 21(a) and (b), the scene surfaces 29, 32, 31, and 34 do acquire the correct model surface labels. However, since the direction vectors associated with all these features are parallel, no unique pose transform can be computed. In such an event, in keeping with the presentation in Section V-D, one of the applicable model surface assignments is given arbitrarily to one of the other scene surfaces, such as scene surface 24. Now a pose transform is computed and its verification made in accordance with the discussion presented earlier.

The recognized objects, in their computed poses, are illustrated in Figs. 1(d) and 19(d); these figures were generated by invoking the PADL system with the computed pose transforms. In the sequence of images in Fig. 2, we show a robot picking up one of these objects. Only rudimentary grasp planning was used for such manipulations. In the model coordinate frame for each object, we identify a set of grasp points and associate with each a gripper transform. This gripper transform is then multiplied by the computed pose transform of the object to figure out the destination transform of the gripper for manipulating the object. Evidently, in some cases the resulting manipulation is not feasible due to the possibility of collisions with either the table or other objects or due to the violation of kinematic constraints. While some of these conditions can be discovered automatically, others require the implementation of rather complex approaches to collision detection and path planning, as has, for example, been implemented in the HANDEY system [34]. Since the focus of this research is not on grasp and path planning, in the experiments reported here, in those cases where the commanded manipulation was not flagged down automatically in the event

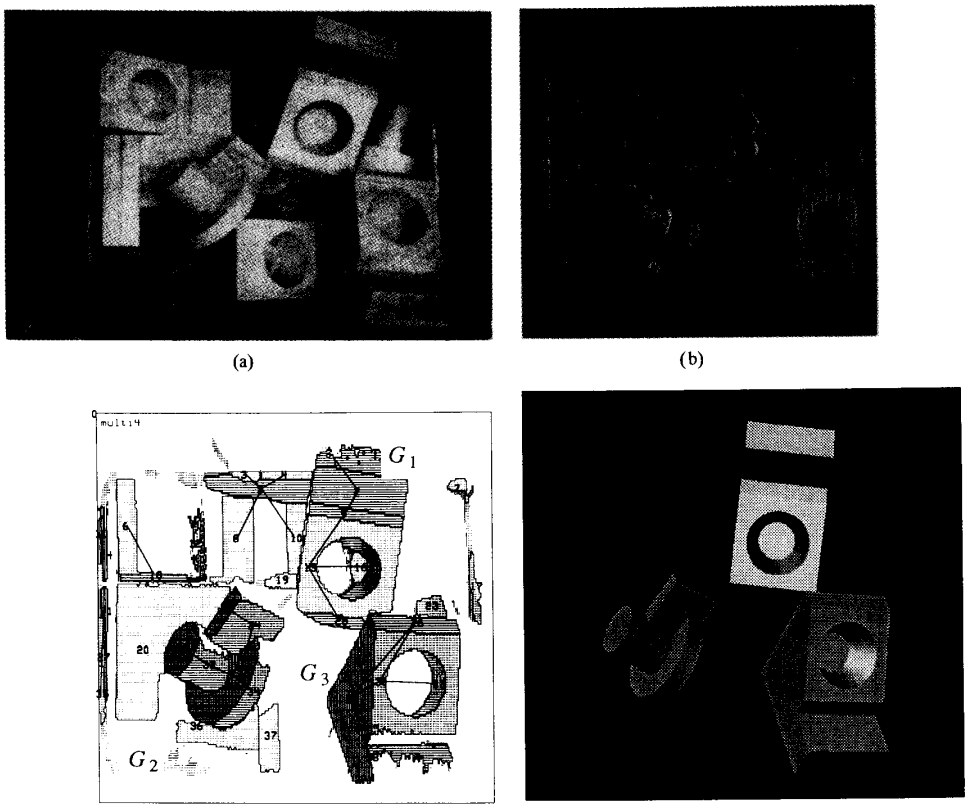



Fig. 19. (a) Multiobject scene. (b) Light-stripe image of the scene made with a robot hand-held structured-light scanner. (c) The preprocessed range map with segmented surfaces and the resulting graph representation. (d) This figure was made by invoking the PADL solid modeling package with the computed identities and poses of some of the objects in the scene of (a).

** Assignment-Consistency Table for G1 **

| | a | b | c | d | e | f | g | h | i | j | k |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1. | . | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | . | 1 | . | . | . | 1 | 1 | 1 | 1 |
| 9 | 1 | 1 | 1 | 1 | . | . | 1 | 1 | 1 | 1 | 1 |
| 13 | 1 | 1 | . | 1 | . | . | . | 1 | 1 | 1 | 1 |
| 15 | . | . | . | . | 1 | . | . | . | . | . | . |
| 16 | 1 | 1 | 1 | 1 | . | 1 | 1 | 1 | 1 | 1 | 1 |
| 25 | 1 | 1 | 1 | 1 | . | 1 | 1 | 1 | 1 | 1 | 1 |

(a)

After Pruning Module

Converged at 3rd iteration

| | a | b | c | d | e | f | g | h | i | j | k |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | . | . | . | . | . | . | 1 | . | . | 1 |
| 2 | . | 1 | . | . | . | . | . | . | . | . | . |
| 9 | . | . | 1 | . | . | . | . | . | . | . | . |
| 13 | . | . | . | 1 | . | . | . | . | . | . | . |
| 15 | . | . | . | . | 1 | . | . | . | . | . | . |
| 16 | . | . | . | . | . | 1 | . | . | . | . | . |
| 25 | . | . | . | . | . | . | 1 | 1 | . | . | |

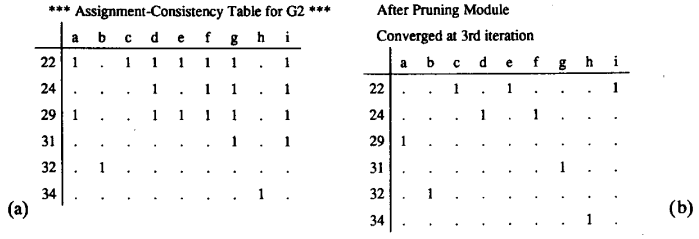
(b)

Fig. 20. (a) The assignment-consistency table produced by the filtering module for scene graph $G_1$ in Fig. 19(c). (b) The assignment-consistency table at the output of the pruning module.

of an impending collision, the robot was brought to a halt manually.

## VII. Conclusion

We believe the computational approach we have presented in this paper is particularly useful when the number of objects in the model library is large and/or when the objects involved possess a large number of surfaces; both these factors lead to large search spaces for object identification and pose estimation. In the flow of control discussed in Section V, the various filters quickly eliminate those models that differ from the scene object in gross details, such as the number and types of surfaces involved, etc. Further pruning of the search space is rapidly accomplished on the basis of relational considerations in the pruning module, which uses a combination of discrete relaxation and bipartite matching. Except for objects of high symmetry, the labeling produced by the pruning module is unique for a sufficient number of scene surfaces to allow us to generate a possible pose transform for the scene object; this transform is subsequently verified by the available labeling of the surfaces that are used for pose-transform calculations. Of course, for objects of high symmetry additional processing has to be invoked since, for such objects, relational considerations invoked by the



*** Assignment-Consistency Table for G2 ***

| | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| 22 | 1 | . | 1 | 1 | 1 | 1 | 1 | . | 1 |
| 24 | . | . | . | 1 | . | 1 | 1 | . | 1 |
| 29 | 1 | . | . | 1 | 1 | 1 | 1 | . | 1 |
| 31 | . | . | . | . | . | . | 1 | . | 1 |
| 32 | . | 1 | . | . | . | . | . | . | . |
| 34 | . | . | . | . | . | . | . | 1 | . |

(a)

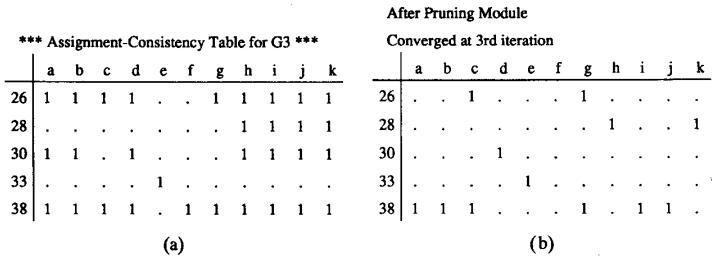
After Pruning Module

Converged at 3rd iteration

| | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| 22 | . | . | 1 | . | 1 | . | . | . | 1 |
| 24 | . | . | . | 1 | . | 1 | . | . | . |
| 29 | 1 | . | . | . | . | . | . | . | . |
| 31 | . | . | . | . | . | . | 1 | . | . |
| 32 | . | 1 | . | . | . | . | . | . | . |
| 34 | . | . | . | . | . | . | . | 1 | . |

(b)

Fig. 21. (a) The assignment-consistency table produced by the filtering module for scene graph $G_2$ in Fig. 19(c). (b) The assignment-consistency table at the output of the pruning module.



*** Assignment-Consistency Table for G3 ***

| | a | b | c | d | e | f | g | h | i | j | k |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 26 | 1 | 1 | 1 | 1 | . | . | 1 | 1 | 1 | 1 | 1 |
| 28 | . | . | . | . | . | . | . | 1 | 1 | 1 | 1 |
| 30 | 1 | 1 | . | 1 | . | . | . | 1 | 1 | 1 | 1 |
| 33 | . | . | . | . | 1 | . | . | . | . | . | . |
| 38 | 1 | 1 | 1 | 1 | . | 1 | 1 | 1 | 1 | 1 | 1 |

(a)

After Pruning Module

Converged at 3rd iteration

| | a | b | c | d | e | f | g | h | i | j | k |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 26 | . | . | 1 | . | . | . | 1 | . | . | . | . |
| 28 | . | . | . | . | . | . | . | 1 | . | . | 1 |
| 30 | . | . | . | 1 | . | . | . | . | . | . | . |
| 33 | . | . | . | . | 1 | . | . | . | . | . | . |
| 38 | 1 | 1 | 1 | . | . | . | 1 | . | 1 | 1 | . |

(b)

Fig. 22. (a) The assignment-consistency table produced by the filtering module for scene graph $G_3$ in Fig. 19(c). (b) The assignment-consistency table at the output of the pruning module.

pruning module can fail to produce a unique pose transform; this additional processing is simple in that it requires that we select from the available labels for a small number of scene surfaces, calculate a pose transform from these labels, and then verify the transform by using the labels available for the other scene surfaces.

We have shown experimental results on single- and multiobject scenes. For multiobject scenes we pointed out the difficulties caused by occlusions. One of these difficulties translated into the partially occluded scene features being declared matchable with a larger number of model features. Also, when occlusions caused some of the intersurface relationships to be missed, the pruning module lost some of its effectiveness, leading to greater backtracking during the pose-transform computation stage and the verification of the transform.

## Appendix A
## Implementation of Bipartite Matching in Polynomial Time

In this Appendix, we will explain how bipartite matching is implemented in polynomial time. Although our exposition is somewhat more tutorial, the algorithm presented here is similar to the one discussed in [37]. Since bipartite matching is so central to the work presented in this paper, we felt compelled to include a discussion on its polynomial time implementation.

For the bipartite matching that is invoked by Filter 3 presented in Section V-B, $X$ will denote the set of nodes in the scene graph and $Y$ the set of nodes in a candidate model graph. On the other hand, when bipartite matching is invoked by the pruning module presented in Section V-C, $X$ will stand for $\Gamma(x)$ and $Y$ for $\Gamma(y)$, where $x$ and $y$ are, respectively, the nodes from the scene and the model graphs, the two nodes being under test for local compatibility. For the latter application of bipartite matching, in accordance with the explanation in Section V-C, we also assume that each node in, say, $\Gamma(x)$ has "absorbed" the attributes of the arc connecting the node with the node $x$. Noting that this is the only difference between the two applications of bipartite matching, in the rest of this Appendix we will simplify the discussion by focusing on just the first application.

For the discussion here, the bipartite graph will be denoted by $G_B = (N_B, A_B)$, where the subscript $B$ is intended to differentiate this graph from model and scene graphs. We will assume that bipartite matching is desired between the two disjoint subsets $X$ and $Y$ that form a partition of $N_B$. Also, the set of arcs $A_B$ is such that every arc $a \in A_B$ connects a node $x \in X$ with a node $y \in Y$ if $x$ and $y$ are node-consistent.[14]

Note that our purpose in bipartite matching is merely to discover whether or not there *exists* a one–one injective mapping from $X$ to $Y$; we do not care about the mapping itself. In other words, through the mechanism of bipartite matching, we want to make sure that for every node in the scene graph there *exists* a distinct node in the model graph—which two nodes from the graphs get paired up is not important. As we will state more formally later, such mappings from $X$ into $Y$ will be called *complete matchings*. Many authors have suggested polynomial time implementations of solutions to the bipartite matching problem [4], [8], [14], [24], [29]. We believe that the oldest of these is the implementation by the Hungarian method [31].[15] A more recent solution to the problem was advanced by Hopcroft and Karp [24] and is based on network flow ideas [8].

Since in our system the nature of the assignment of a model node to a scene node is binary in nature, the matching algorithms most relevant to us are those for the Hungarian method and

[14] At the risk of sounding repetitive, we must emphasize the fact that the nature of the arcs here bears no relationship to the arcs in the graphs for scene or model representation. For the bipartite matching invoked by Filter 3, two nodes are connected by an arc whenever the nodes satisfy the conditions for node consistency stated in Section IV. For the bipartite matching invoked by the pruning module, a scene node is connected by arc to a model node provided there is a "1" in the corresponding position in the assignment-consistency table and provided the "absorbed" arc attributes are the same, as per the discussion in Section V-C.

[15] In a somewhat more general problem formulation, when weights are assigned to the different possible pairing from $X$ and $Y$, we may not insist on a complete matching, but an optimal matching that maximizes the sum of the weights. Such a problem may be solved by the Kuhn–Munkres method [4].

the one proposed by Hopcroft and Karp. For reasons having to do with the programming effort required, we chose to start with the Hungarian method, even though the complexity of this method, being $O(\min(|X|,|Y|)\cdot|A_B|)$, exceeds that of the method proposed by Hopcroft and Karp, the time complexity of the latter method being $O(\sqrt{(|X|+|Y|)}\cdot|A_B|)$. For our application, if we assume that $X$ corresponds to scene graph nodes, with the understanding that only a single object is present in the scene, and $Y$ to model graph nodes $|X| < |Y|$, the complexity of the Hungarian method becomes $O(|X|\cdot|A_B|)$. It is important to realize that the worst-case time complexity of the Hungarian method is not that different from that of the Hopcroft and Karp algorithm, since in the worst case each node in $X$ will be consistent with every node in $Y$, implying $A_B = |X||Y|$. We use the phrase "worst case" for only discussing the comparative behavior of the two algorithms; of course, if each node in the scene graph is consistent with every node in the model graph, it would be trivial to discover a complete matching.

We need to define a few terms before we present the algorithm.

- A *match* $M$ is a list of node-consistent ordered pairs $\{(x_i,y_j)\}$, $x_i \in X$, $y_j \in Y$, such that $(x_i,y_j) \in A_B$. Clearly, $M \subset A_B$, with the property that no two edges in $M$ share the same node. In the bipartite graph shown in Fig. 23(a) the set of arcs $\{(1,c),\ (3,a)\ \cdots\}$ constitutes a matching $M$. Note that this $M$ is not unique.
- A node $x_i$ or $y_j$ participating in $M$ is called *M-saturated;* otherwise, a node of $G_B$ is *M-unsaturated* or just *exposed.* Given the match $M$ in Fig. 23(b), in the bipartite graph of Fig. 23(a) the nodes in the sets $\{1,2\}$ and $\{a,b\}$ are $M$-saturated, whereas the nodes in the sets $\{3,4\}$ and $\{c,d\}$ are $M$-unsaturated.
- An *M-alternating path* in $G_B$ is a path whose arcs are alternately in $M$ and $A_B - M$. The set of arcs $\{(3,a),\ (a,1),\ (1,c)\}$ displayed in Fig. 23(c) constitutes an $M$-alternating path with respect to the $M$ shown in Fig. 23(b) and for the bipartite graph in Fig. 23(a).
- An *M-augmenting path* is an $M$-alternating path whose origin and terminus are $M$-unsaturated. The $M$-alternating path in Fig. 23(c) is also an $M$-augmenting path.
- For any set $S$ of nodes in $X$, we denote by $N(S)$ the set of nodes from $Y$ that are in node consistency with the nodes in $S$. For example, for the graph of Fig. 23(a), for the nodes $\{2,4\}$ in $X$, $N(\{2,4\})$ is given by $\{a,b,c\}$.
- During the process of pairing up consistent nodes from $X$ and $Y$ for the purpose of constructing a match $M$, we may reach a point where no further pairings are possible. At that point, $M$ is called a *maximal* match. In other words, a *maximum matching* is a matching $M$ to which no further arcs from $A_B$ can be added. For example, the matching $\{(1,a),(2,b)\}$ is a maximal matching for the $G_B$ shown in Fig. 23(a); this maximal matching is pictorially depicted in Fig. 23(b). No further arcs from $A_B$ shown in Fig. 23(a) could be added to this $M$ without violating the one–one injective mapping condition on $M$.

It is most important to realize that a given maximal matching may not posses maximum cardinality; in other words, there may exist maximal matchings of higher cardinality. Again for the case of $G_B$ shown in Fig. 23(a), the maximal matching $\{(1,c),\ (2,b),\ (3,a)\}$ is of higher cardinality than the maximal matching $\{(1,a),\ (2,b)\}$. A necessary and sufficient condition for a maximal matching to be of maximum cardinality is given by the following theorem, whose proof can be found in [4].

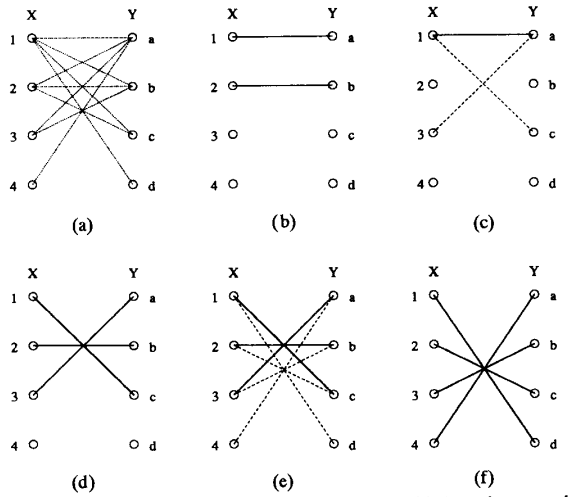



Fig. 23. Procedure for finding a maximal matching. (a) Arcs denote node consistency. (b) An initial maximal matching $M_0$. (c) An augmenting path $P_1$ with respect to the maximal matching $M_0$ starting from node 3. (d) The maximal matching, $M_1$, obtained by applying the EXCLUSIVE-OR operator to the arcs in (b) and (c). $M_1 = M_0 \oplus P_1$. (e) An augmenting path $P_2$ with respect to the matching $M_1$ starting at node 4. (f) The maximal matching $M_2$ obtained by applying the EXCLUSIVE-OR operator to the arcs in (d) and (e). $M_2 = M_1 \oplus P_2$.

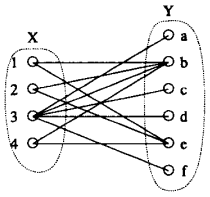



Fig. 24. Bipartite graph in which a complete matching cannot exist.

*Theorem 1:* A matching $M$ in $G_B$ is the *largest maximal matching* if and only if $G_B$ contains no $M$-augmenting path.

We will now introduce two more characterizations of matchings.

- A matching is *perfect* when every node in $G_B$ is $M$-saturated. Clearly, unless $|X| = |Y|$ is true, a matching will not be perfect.
- When we match scene graphs with model graphs, the condition $|X| \le |Y|$ is usually satisfied, if we assume that $X$ represents the nodes in the scene graph and $Y$ the nodes in the model graph. In such bipartite graphs, the notion of *complete matching* becomes important. A matching $M$ is complete when every node in $X$ is $M$-saturated.

A necessary and sufficient condition for the existence of a complete matching in a bipartite graph was given by Hall [4] and is presented below without proof.

*Theorem 2:* $G_B$ contains a complete matching if and only if

$$|N(S)| \ge |S| \text{ for all } S \subseteq X.$$

This theorem says that a complete matching of $X$ into $Y$ exists if and only if every subset $S$ of the nodes in $X$ is "collectively connected" to $|S|$ or more nodes in $Y$. This condition is not satisfied in Fig. 24 because the subset $\{1,2,4\}$ from $X$ is connected

to $\{b, e\}$, a subset from $Y$ whose cardinality is lower than that of the subset from $X$.

While the condition of the above theorem sounds simple, its implementation directly as stated is not, since that would require exponential time, there being $2^{|X|}-1$ nonempty subsets in $X$ and the condition must be tested for each subset [7]. We presented Theorem 2 not because it leads to any useful computational strategies, but because it states the most well-known condition for the existence of a complete matching in a bipartite graph and for that reason is the method most workers think of implementing when first getting started with the notions under discussion here. A computationally efficient way to discover the existence of a complete matching is via the previously mentioned Hungarian method and the Hopcroft and Karp algorithm, both intended for finding the largest maximal matching in a bipartite graph. Clearly, if the cardinality of the maximal matching found by one of the procedures is less than $|X|$, we do not have a complete matching.

This brings us to the heart of this Appendix: the algorithm for finding a complete matching in a bipartite graph. The algorithm is based on the lemma that if $M$ is a matching and $P$ an augmenting path relative to $M$, then by combining $M$ and $P$ via an EXCLUSIVE-OR operation we obtain another matching, denoted by $M \oplus P$, whose cardinality exceeds that of $M$ by one. Two important points to note about this lemma are 1) the path $P$ does not have to be the shortest possible path, and 2) the matching $M$ does not have to be a maximal matching. In our use of the lemma in the procedure presented below, we will only deal with maximal matchings.

The algorithm is iterative at the top level. It starts out with an initially constructed maximal matching, which is then refined in subsequent iterations, until either a complete matching is discovered or failure declared. Of course, a separate procedure must be implemented to discover an initial maximal matching; this procedure can be based on a direct comparison of each $X$ node with those $Y$ nodes that have not been already matched, until no more $X$ nodes can be paired up.

The algorithm for complete matching may now be described by the following steps. We will assume that we have available to us a maximal matching already and that we now want to discover a maximal matching of larger cardinality; we terminate the process when the cardinality of the maximal matching found equals that of $X$.

STEP 1: If the cardinality of the current maximal matching $M$ is equal to $|X|$, we are done; otherwise go to STEP 2.

STEP 2: Select an exposed node in $X$ and starting from this node construct an $M$-augmenting path that alternately uses arcs from $A_B - M$ and $M$. As was shown in [37], the complexity of this operation is $O(|A_B|)$. This measure of complexity is based on the fact that we can select any exposed node from $X$ and, if we do not find an augmenting path starting from that node, then there simply does not exist an augmenting path with respect to the given matching. An algorithm for finding an augmenting path with respect to a given matching is described in [4] and [37]. If an augmenting path cannot be found, declare failure.

STEP 3: Form a new maximal matching by combining the current maximal matching and the augmenting path found in the previous step via an EXCLUSIVE-OR operation. By EXCLUSIVE-OR operation we mean that we eliminate those arcs that are in both the matching and the augmenting path.

STEP 4: Go to Step 2.

## Appendix B
## Calculation of the Pose Transform

It is interesting to note that when the pose transform $\boldsymbol{T}$ is expressed as a $4 \times 4$ homogeneous matrix, at least theoretically the elements of the matrix could be computed by a knowledge of the coordinates of three noncollinear points on the object in the scene and their corresponding locations in the model; these three points could, for example, be vertices, centroids of surfaces, etc. Whereas the three points translate into nine equations, six additional equations are generated by the orthonormality constraint on the $3 \times 3$ upper left submatrix of $\boldsymbol{T}$ which corresponds to rotation. This constraint implies that each column of the submatrix be of unit magnitude and orthogonal to the other two columns. These 15 equations can be used to solve for only 12 unknowns since any solutions must also obey rigid body constraints which say the distances and/or angles between the points must not change when the object is tranformed by $\boldsymbol{T}$; that is, if $\boldsymbol{p}_s^1$, $\boldsymbol{p}_s^2$, and $\boldsymbol{p}_s^3$ are the three scene points and $\boldsymbol{p}_m^1$, $\boldsymbol{p}_m^2$, and $\boldsymbol{p}_m^3$ their model counterparts, the rigid body constraints could take the form of three equalities like $|\boldsymbol{p}_s^i - \boldsymbol{p}_s^j| = |\boldsymbol{p}_m^i - \boldsymbol{p}_m^j|$. As it turns out, 12 equations are sufficient to compute $\boldsymbol{T}$ since its last row will always have elements that are known already for rigid body transformations. While, owing to its straightforward nature, this approach to the computation of $\boldsymbol{T}$ is attractive, it does suffer from two difficulties. The first difficulty has to do with the fact that in the presence of occlusion it may be impossible to precisely locate points such as the centroids of surfaces, etc. The second difficulty has to do with the estimation of the components of $\boldsymbol{T}$ in the presence of noise, since in that case the solutions must be obtained by some sort of optimization in a fairly high-dimensional space spanned by the equations.

For these reasons, the pose-transform matrix is estimated most frequently by first decomposing $\boldsymbol{T}$ into its rotational and translational parts and then estimating each separately. Separate computation of the two parts is justified particularly by the fact that the rotational part can be computed more robustly from directional vectors, such as surface normals, directions of axes of cylinders, etc., leaving the problem of estimation of the translational part, for which a single point will suffice. Unfortunately, in practice, in the presence of occlusions, it is often difficult to reliably locate even one *known* point for the computation of the translational part and one must resort to the method discussed later in this Appendix. It is important to bear in mind that the decomposition of a pose-transform matrix into rotational and translational parts is not unique in general. As a matter of fact, there are infinite combinations of a rotation followed by a translation or *vice versa* that would take a rigid body from one pose into another. However, if we constrain the axis of rotation to always pass through the origin of the coordinate system, then the decomposition of $\boldsymbol{T}$ is unique.

Let $\boldsymbol{R}$ and $\boldsymbol{t}$ be, respectively, the rotational and the translational parts of the pose transform. Then given a directional vector $\boldsymbol{v}_s$ from the scene and its correspondent $\boldsymbol{v}_m$ from the model, we have

$$\boldsymbol{R} \cdot \boldsymbol{v}_s^i = \boldsymbol{v}_m^i, \quad i = 1, \cdots, N. \tag{B1}$$

Similarly, if we have established correspondence between a point $\boldsymbol{p}_s$ in the scene and a point $\boldsymbol{p}_m$ on the model, we have

$$\boldsymbol{R} \cdot \boldsymbol{p}_s + \boldsymbol{t} = \boldsymbol{p}_m. \tag{B2}$$

We will now discuss the procedures used for the computation of $\boldsymbol{R}$ and $\boldsymbol{t}$.

### A. Computation of the Rotation Matrix

Given a set of directional vectors $\boldsymbol{v}_s^1, \boldsymbol{v}_s^2, \cdots, \boldsymbol{v}_s^N$, their correspondents $\boldsymbol{v}_m^1, \boldsymbol{v}_m^2, \cdots, \boldsymbol{v}_m^N$, and the following set of relationships between each pair of correspondents:

$$\boldsymbol{R} \cdot \boldsymbol{v}_s^i = \boldsymbol{v}_m^i, \tag{B3}$$

we want to estimate an $\boldsymbol{R}$ that minimizes

$$\sum_{i=1}^{N} |\boldsymbol{R} \cdot \boldsymbol{v}_s^i - \boldsymbol{v}_m^i|^2. \tag{B4}$$

One might be tempted to solve this minimization problem by first casting the $N$ equations that are represented by (B1) into the following composite form:

$$\boldsymbol{R}[\boldsymbol{v}_s^1, \boldsymbol{v}_s^2, \cdots,] = [\boldsymbol{v}_m^1, \boldsymbol{v}_m^2 \cdots,] \tag{B5}$$

which is more compactly represented by

$$\boldsymbol{R}\boldsymbol{V}_s = \boldsymbol{V}_m \tag{B6}$$

where $\boldsymbol{V}_s$ and $\boldsymbol{V}_m$ are $3 \times N$ matrices, and then invoking the pseudo-inverse approach for solving such equations to yield for the solution

$$\boldsymbol{R} = \boldsymbol{V}_m \boldsymbol{V}_s^T [\boldsymbol{V}_s \boldsymbol{V}_s^T]^{-1}. \tag{B7}$$

where the superscript $T$ means the transpose operation. Unfortunately, such a solution will, in general, prove to be erroneous for the following reason. The solution in (B7) could be considered to be a consequence of setting the following derivative to zero:

$$\frac{\delta}{\delta \boldsymbol{R}} (\boldsymbol{V}_m - \boldsymbol{R}\boldsymbol{V}_s)(\boldsymbol{V}_m - \boldsymbol{R}\boldsymbol{V}_s)^T = 0. \tag{B8}$$

However, the minimization implied by setting the derivative shown to zero makes no intuitive sense because the argument given to the derivative operator is not a scalar error metric but a $3 \times 3$ matrix. Clearly, we cannot expect to obtain an $\boldsymbol{R}$ that would simultaneously minimize all nice elements of this matrix. One could get around this hurdle by recasting (B6) in a matrix–vector form by using, for example, a lexicographic recoding to express $\boldsymbol{R}$ as a vector. We would now be able to use a single error metric and the resulting solution obtained by the pseudo-inverse approach would represent an attempt at a minimization of this error metric with respect to the nine components of $\boldsymbol{R}$. But, such a minimization is really not valid since the nine components of $\boldsymbol{R}$ are not independent; as a matter of fact, they are highly interdependent considering that $\boldsymbol{R}$ possesses only three degrees of freedom.

Several approaches have been advanced for solving the equations in (B3) for $\boldsymbol{R}$ [1], [5], [6], [9], [11], [12], [17]. Grimson and Lozano-Perez's approach [17] is based on the notion that given a pair of correspondents $(\boldsymbol{v}_s^i, \boldsymbol{v}_m^i)$, the rotational axis must lie on the perpendicular bisector plane of the line joining $\boldsymbol{v}_s^i$ and $\boldsymbol{v}_m^i$. For example, in Fig. 25 the axis for the rotation that takes $\boldsymbol{v}_s^i$ into $\boldsymbol{v}_m^i$ must lie on the plane denoted by $\boldsymbol{L}^i$. Therefore, given two pairs of correspondents $(\boldsymbol{v}_s^i, \boldsymbol{v}_m^i)$ and $(\boldsymbol{v}_s^j, \boldsymbol{v}_m^j)$, we can find the axis of rotation as the line corresponding to the intersection of the two bisecting planes, as demonstrated by the intersections of the two planes $\boldsymbol{L}^i$ and $\boldsymbol{L}^j$ in Fig. 25. This amounts to setting the axis of rotation to

$$(\boldsymbol{v}_s^i - \boldsymbol{v}_m^i) \times (\boldsymbol{v}_s^j - \boldsymbol{v}_m^j) \tag{B9}$$

for two pairs of corresponding directional vectors. This scheme obviously will not work when the two difference vectors $\boldsymbol{v}_s^i - \boldsymbol{v}_m^i$ and $\boldsymbol{v}_s^j - \boldsymbol{v}_m^j$ are parallel, a not unlikely situation to arise in practice. As shown in Fig. 26, the rotation of the cube gives rise to $(\boldsymbol{v}_s^1, \boldsymbol{v}_m^1)$ and $(\boldsymbol{v}_s^2, \boldsymbol{v}_m^2)$ which leads to exactly the situation where the two difference vectors are parallel and not usable for the calculation of the axis of rotation. This approach also cannot be used for a scene directional vector if it is parallel to the corresponding model directional vector. In the method of Grimson and Lozano-Perez, when more than two scene directional vectors from a scene are available, the axis of rotation is computed for each pair of them and the result averaged.

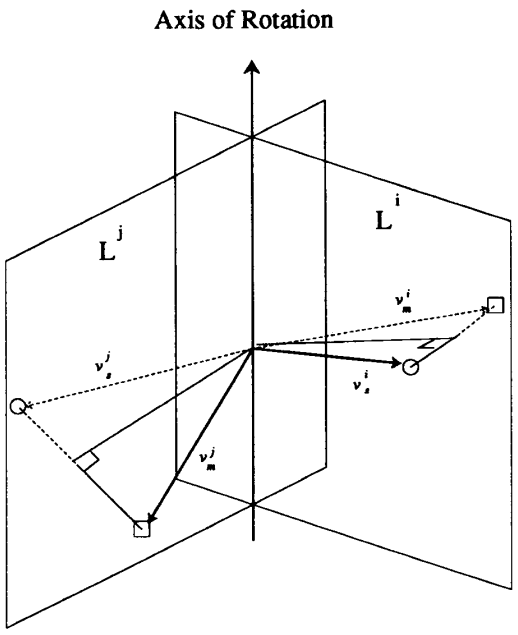



Fig. 25. If the model directional vector $v_m^j$ corresponds to the scene directional vector $v_s^j$, then the axis of rotation of the object must lie on the bisecting plane $\boldsymbol{L}^j$. The intersection of two such bisecting planes $\boldsymbol{L}^i$ and $\boldsymbol{L}^j$ defines the axis of rotation.

Fan, *et al.*'s approach [9] is identical to the one by Grimson and Lozano-Perez except for the final computation of the axis of rotation from multiple pairs of scene directional vectors. Instead of just averaging the results for each pair of scene directional vectors, they compute the sum of the angles between the axis of rotation for the pair and the axes of rotation for all other pairs. They retain that axis of rotation which leads to the smallest value for this sum. In the approach advanced by Arun *et al.* [1], a set of corresponding points from the scene and the model is used for computing first the rotation matrix and then the translation vector. If this approach were to be used by us, the scene object would first be rotated around the mean of all the points to be used for the calculation of the pose transform, and the rotation would be such as to make the orientation of the scene object similar to the orientation of the model object. The translational component is then computed by bringing into coincidence by a translation the rotated mean of the scene points and the mean of the model points. This approach is not suitable for our application for one of the reasons mentioned at the beginning of this Appendix regarding the use of points for the computation of a pose transform: the difficulty associated with correctly identifying and locating point features in the presence of occlusions.

The approach we have used in our work is based on the derivation originally advanced by Faugeras and Hebert [11], [12] and subsequently used in our lab in the work reported in [6].[16] This derivation uses the quaternion approach to minimize the sum of the squared errors between the rotated scene directional vectors and the corresponding model directional vectors by solving the following equations:

$$\frac{\delta}{\delta \boldsymbol{R}} \sum_{i=1}^{N} |\boldsymbol{R} \cdot \boldsymbol{v}_s^i - \boldsymbol{v}_m^i|^2 = 0. \tag{B10}$$

A quaternion is a 4-tuple, frequently expressed by a combination of a scalar and a 3-D vector. For example, as shown in [6] and [12], a rotation through angle $\theta$ around an axis whose direction is given by the unit vector $\boldsymbol{a}$ is given by the following quaternion:

$$\boldsymbol{Q}_R = \left(\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\boldsymbol{a}\right). \tag{B11}$$

Now, an ordinary directional vector $\boldsymbol{v}$ would be represented in the quaternion form as $(0, \boldsymbol{v})$, and its rotation by $\boldsymbol{Q}_R$ would result in the quaternion $(0, \boldsymbol{R} \cdot \boldsymbol{v})$, where

$$(0, \boldsymbol{R} \cdot \boldsymbol{v}) = \boldsymbol{Q}_R * (0, \boldsymbol{v}) * \overline{\boldsymbol{Q}}_R. \tag{B12}$$

The symbol "*" here denotes a multiplication between two quaternions, as defined by the following expression for two quaternions, $\boldsymbol{Q} = (\theta, \boldsymbol{a})$ and $\boldsymbol{Q}' = (\theta', \boldsymbol{a}')$:

$$\boldsymbol{Q} * \boldsymbol{Q}' = (\theta\theta' - \boldsymbol{a} \cdot \boldsymbol{a}', \boldsymbol{a} \times \boldsymbol{a}' + \theta\boldsymbol{a}' + \theta'\boldsymbol{a}) \tag{B13}$$

By substituting quaternions for the various quantities in the left-hand side of (B10), it can be shown that that equation is identical to

$$\frac{\delta}{\delta \boldsymbol{R}}(\boldsymbol{Q}_R \cdot \boldsymbol{A} \cdot \boldsymbol{Q}_R^T) = 0 \tag{B14}$$

where $\boldsymbol{A}$ is given by

$$\boldsymbol{A} = \sum_{i=1}^{N} \boldsymbol{B}_i \boldsymbol{B}_i^T \tag{B15}$$

where

$$\boldsymbol{B} = \begin{bmatrix} 0 & -c_x^i & -c_y^i & -c_z^i \\ c_x^i & 0 & b_z^i & -b_y^i \\ c_y^i & -b_z^i & 0 & b_x^i \\ c_z^i & b_y^i & -b_x^i & 0 \end{bmatrix} \tag{B16}$$

and

$$\begin{aligned} \boldsymbol{b}^i &= \boldsymbol{v}_s^i + \boldsymbol{v}_m^i \\ \boldsymbol{c}^i &= \boldsymbol{v}_s^i - \boldsymbol{v}_m^i. \end{aligned} \tag{B17}$$

The quaternion $\boldsymbol{Q}_R$ that minimizes the argument of the derivative operator in (B14) is an eigenvector of matrix $\boldsymbol{A}$. If we denote an eigenvector of $\boldsymbol{A}$ by the 4-tuple $(\alpha, \beta, \gamma, \delta)$, then it follows from (B11) that the rotation angle $\theta$ associated with the rotational transform is given by

$$\theta = 2\cos^{-1}(\alpha). \tag{B18}$$

Again, from (B11), the axis of rotation would be given by

$$\boldsymbol{a} = (\beta, \gamma, \delta) \Big/ \sin\left(\frac{\theta}{2}\right). \tag{B19}$$

[16] Reference [6] also contains a more tutorial rederivation of the result derived originally by Faugeras and Hebert.

These calculations for $\theta$ and $\boldsymbol{a}$ are made under the constraint that the magnitude of the eigenvector $(\alpha, \beta, \gamma, \delta)$ is normalized to unity. Also, the eigenvector chosen for the calculation of $\theta$ and $\boldsymbol{a}$ must correspond to the minimum eigenvalue.

An important question from the standpoint of implementation is: What is the least number of directional vectors required for the computation of $\boldsymbol{R}$? For each $i$, (B3) really gives us not three equations, but only two, since the directional vectors are unit vectors and, therefore, one of their components can be predicted from the other two. Since $\boldsymbol{R}$ contains three independent variables, two for the unit vector representing the axis of rotation and one for the angle of rotation around this unit vector, we therefore need at least two equations of the type in (B3) for the computation of $\boldsymbol{R}$. As was mentioned in [6], two equations like (B3) do not really translate into four independent equations for the three unknowns in $\boldsymbol{R}$ since the directional vectors must also satisfy the rigid body constraint $\boldsymbol{v}_s^i \cdot \boldsymbol{v}_s^j = \boldsymbol{v}_m^i \cdot \boldsymbol{v}_m^j$.

### *B. Computation of the Translation Vector*

If it is possible to identify at least one point correctly in the scene, knowledge of the rotation matrix can be used in the following equation, obtained from (B2), to easily compute the translational component of the pose transform:

$$\boldsymbol{t} = \boldsymbol{p}_m - \boldsymbol{R} \cdot \boldsymbol{p}_s \tag{B20}$$

where $\boldsymbol{p}_s$ is the point in the scene and $\boldsymbol{p}_m$ its correspondent on the model. Of course, as was done in [6], if it is possible to identify more than one such point in a scene, the vector $\boldsymbol{t}$ can be computed by minimizing the following error norm:

$$\sum_{i=1}^{N} |\boldsymbol{p}_m^i - \boldsymbol{R} \cdot \boldsymbol{p}_s^i - \boldsymbol{t}|^2 \tag{B21}$$

where $\boldsymbol{p}_s^1, \boldsymbol{p}_s^2, \cdots, \boldsymbol{p}_s^N$ are the point features from the scene and $\boldsymbol{p}_m^1, \boldsymbol{p}_m^2, \cdots, \boldsymbol{p}_m^N$ the corresponding point features from the model. Minimizing this expression by taking a derivative with respect to $\boldsymbol{t}$ and setting the derivative to 0 yields the following solution for $\boldsymbol{t}$:

$$\boldsymbol{t} = \sum_{i=1}^{N} \boldsymbol{p}_m^i - \boldsymbol{R} \cdot \sum_{i=1}^{N} \boldsymbol{p}_s^i. \tag{B22}$$

As the reader can see, the optimum solution for $\boldsymbol{t}$ is nothing more than a translation of the rotated mean vector associated with the scene points to the mean vector associated with the model points.

In dealing with scenes containing occlusions, often it is not possible to identify any robustly detectable point features at all with an object. By robustly detectable point features we mean points such as vertices where three or more planar surfaces come together, apexes of conical surfaces, etc. There are other point features that one could use, such as centroids of planar surfaces, etc., but their identification and location is more prone to the deleterious effects of occlusion. For these reasons, one would like to use nonpoint features for the estimation of the translation vector. Following Faugeras and Hebert [11], [12], we will now present a method for doing so.

The pose estimation formalism of Faugeras and Hebert uses visible portions of surfaces to set up an error metric whose minimization leads to the best possible solution of the translational vector. The rationale behind this approach is best explained with the help of Fig. 27. Let's say that $AB$ is the visible fragment of a scene surface whose model correspondent is the surface $CD$. To compute the translational vector $\boldsymbol{t}$, we first rotate the scene

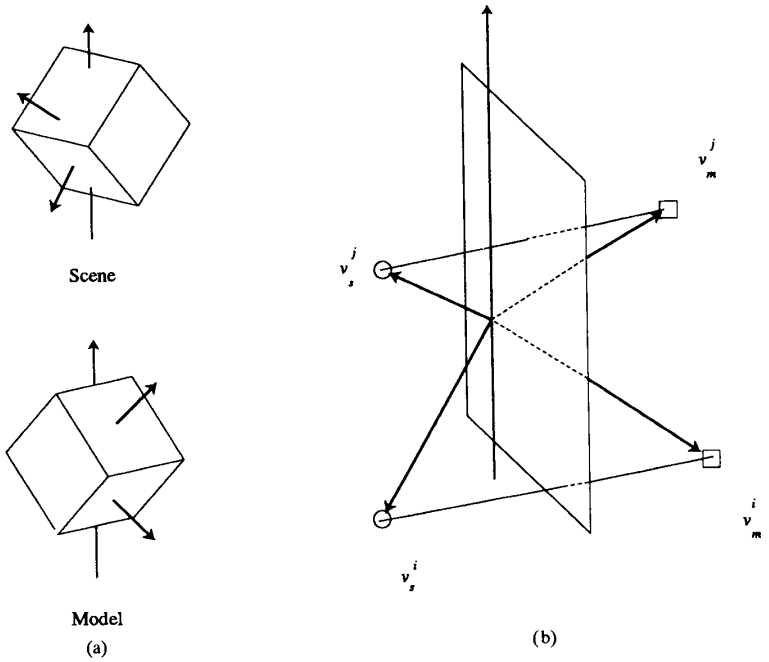



Fig. 26. (a) A model object and the corresponding scene object rotated about the vertical axis. (b) Although there exists a unique axis of rotation for this case, one would not be able to compute it by using (B9) because the difference vectors $v_s^i - v_m^i$ and $v_s^j - v_m^j$ are parallel.

surface $AB$ by the rotation matrix $\boldsymbol{R}$ around the origin of the coordinate system, the result being the surface fragment $A'B'$. We now construct a vector, denoted by $t_c$, from the centroid of the model surface $CD$ to the centroid of $A'B'$ and then project this vector onto the surface normal to $CD$. This projection is equal to perpendicular distance $PQ$ between the rotated scene surface and the model surface. The length of $PQ$ is given by

$$\overline{PQ} = \boldsymbol{t}_c \cdot \boldsymbol{v}_m \tag{B23}$$

where $v_m$ is normal to the model surface. If $p_m$ and $p_s$ denote, respectively, the centroids of the model surface and the surface fragment in the scene, the equations of the two surfaces are given by

$$\begin{aligned} \boldsymbol{p}_m \cdot \boldsymbol{v}_m &= d_m \\ \boldsymbol{p}_s \cdot \boldsymbol{v}_s &= d_s \end{aligned} \tag{B24}$$

where $v_s$ is the measured normal to the scene surface and $d_m$ and $d_s$ are the perpendicular offset distances to the two surfaces. It is clear that if the scene surface is rotated around the origin so that its orientation is the same as that of the model surface, the distance $\overline{PQ}$ should equal

$$\overline{PQ} = d_m - d_s. \tag{B25}$$

By comparing (B23) and (B25), we obtain

$$\boldsymbol{t}_c \cdot \boldsymbol{v}_m = d_m - d_c. \tag{B26}$$

As the occlusion of the surface of which $AB$ is a fragment is varied, the location of the centroid of what is visible will also change, leading to a different $t_c$. However, regardless of occlusion, all $t_c$'s will obey the above equation. Therefore, it stands to reason that if we have many planar surfaces in a scene belonging to the same object, we should seek a $t$ which minimizes the following error criterion:

$$\sum_{i=1}^{N} \left(\boldsymbol{t} \cdot \boldsymbol{v}_m^i - \left(d_m^i - d_s^i\right)\right)^2 \tag{B27}$$

where superscript $i$ refers to the $i$th planar surface in the scene and its model correspondent. The solution to this minimization is given by

$$\boldsymbol{t} = \boldsymbol{D}\boldsymbol{V}_m^T\left(\boldsymbol{V}_m\boldsymbol{V}_m^T\right)^{-1} \tag{B28}$$

where $\boldsymbol{V}_m$ is the $3 \times N$ matrix

$$\boldsymbol{V}_m = \left[\boldsymbol{v}_m^i, \boldsymbol{v}_m^2, \cdots, \boldsymbol{v}_m^N\right]$$

and $\boldsymbol{D}$ is the column vector

$$\boldsymbol{D} = \left[d_s^1 - d_m^1, d_s^2 - d_m^2, \cdots, d_s^N - d_m^N\right].$$

Clearly, this method for the determination of the translational vector can only be applied to a collection of planar surfaces on an object. For nonplanar surfaces, one has no choice but to resort to point feature based methods discussed earlier. If no distinguished points, such as vertices, can be identified, in some cases it is possible to use imaginary points formed by intersecting the axes of cylindrical or conical surfaces with the planes representing the flat bases on which such nonplanar surfaces might be mounted.

### C. Testing Direction Vectors for Sufficient Information

The scene surfaces chosen for the minimization of the error metric in (B27) should be such that their orientation vectors, as given by $v_s^1, v_s^2, \cdots, v_s^N$ are not all parallel. When these directional vectors are parallel, so will be their model correspondents $v_m^1, v_m^2, \cdots, v_m^N$; as a result there will not exist a unique translational vector $t$ for the object. Fig. 28 illustrates this

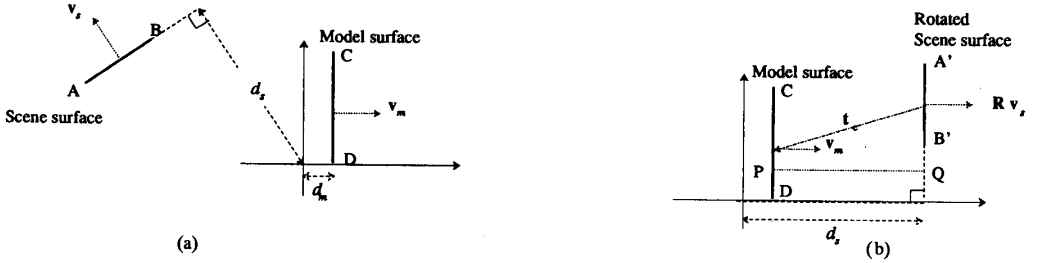



Fig. 27. (a) $AB$ represents the visible fragment of a surface in the scene and $CD$ the model correspondent of the surface. The distances $d_s$ and $d_m$ are, respectively, the perpendicular distances of the two surfaces from the origin. (b) To compute the translational component of the pose transform, we first rotate $AB$ in such a manner that its orientation becomes identical to that of $CD$, as shown here.

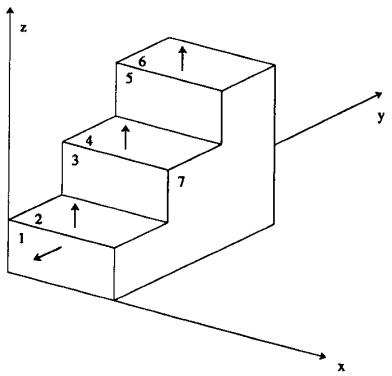


Fig. 28. There is not sufficient information contained in surfaces 2, 4, and 6 of this staircase object for computing the translational component of the pose transform.

point. If we only use the surfaces 2, 4, and 6 in the minimization in (B27), the translational vector will only be constrained with respect to the $z$ axis of the coordinate frame. In other words, there will not be sufficient information in (B27) for the computation of the other two components of $\boldsymbol{t}$. Also, taking surface 1 into account will provide us with another constraint but will still leave $\boldsymbol{t}$ unspecified with respect to one degree of freedom.

It is clear that before using a set of surfaces for the computation of the translational vector, we must examine them for sufficiency of information. One simple approach for doing this would consist of taking triple scalar products for all combinations of three directional vectors and insisting that for at least one combination the product be nonzero. We have not used this approach.

We will now present our approach for ascertaining that there is sufficient directional information contained in the directional vectors. Basically, we want to make sure that the dimensionality of the space spanned by the directional vectors is three. In matrix algebra, this amounts to saying that if we construct a matrix whose rows are the directional vectors available to us, then the rank of this matrix should equal 3.[17] The rank of such a matrix could easily be obtained by counting the number of nonzero rows in a Gauss-reduced form or the row-echelon form of the matrix, since these nonzero rows constitute basis vectors of the space spanned by the directional vectors. However with noisy directional vectors, this method does not yield reliable results.

Our strategy is based on the fact that if $\boldsymbol{V}$ denotes the matrix whose rows are the directional vectors, then the rank of $\boldsymbol{V}$ is the same as the rank of $\boldsymbol{V}^T\boldsymbol{V}$. Since $\boldsymbol{V}^T\boldsymbol{V}$ is a square matrix of size $3 \times 3$, its determinant exists and will be nonzero only if the rank equals three. Of course, in the presence of noise the determinant may not equal zero even when the rank is less than three, so it is best to use a threshold for making this test. In using this method, one must bear in mind that the determinant is the product of the eigenvalues of the matrix $\boldsymbol{V}^T\boldsymbol{V}$ and that each eigenvalue represents the "support" for the corresponding eigenvector in the directional vectors data. In other words, if a majority of the directional vectors are lined up with, say, the $z$ axis, then one of the eigenvectors would be nearly parallel to the $z$ axis and its eigenvalue would take a value much larger than the other two. Therefore, even when we have directional vectors spanning 3-space, the product of the three eigenvalues can be small if the "support" for one or two of the basis vectors is relatively low in comparison to the support for the rest.

In our implementation, we have used a slight variation on this method of checking the value of the determinant of $\boldsymbol{V}^T\boldsymbol{V}$. We carry out an eigenvector analysis of the $\boldsymbol{V}^T\boldsymbol{V}$ matrix and we then examine the eigenvalues individually. If all three eigenvalues exceed an acceptance threshold, then the rank of $\boldsymbol{V}^T\boldsymbol{V}$ is clearly three and we are done. On the other hand, if only two eigenvalues exceed the acceptance threshold, then we do not have sufficient information for the computation of $\boldsymbol{t}$ using (B27). However, we can still use the directional vectors for the computation of $\boldsymbol{R}$ and, in some cases, depend upon other methods for the computation of $\boldsymbol{T}$, one of these other methods being the calculation of $\boldsymbol{T}$ by using centroids of surfaces, especially if by comparing areas it can be determined that the extent of occlusion is small. In other cases, it might be possible to use the centroids of cylindrical surfaces in a similar manner, such centroids being defined as points on the axes of cylinders, as was done in Section V-D.

[17] The reader should note that for a nonsquare matrix, the maximum rank cannot exceed the minimum of the number of rows or columns.

## Acknowledgment

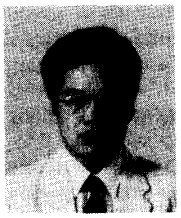
The authors wish to express our appreciation to all members of the Robot Vision Lab for contributing to the intellectually stimulating environment without which the work reported here might not have been possible. Hardware support was provided by M. Carroll and software support by J. Lewis, both research engineers in the Robot Vision Lab; we owe them many thanks. We also owe thanks to C.-H. Chen for many valuable discussions regarding 3-D object recognition and also his software on calibration and range data preprocessing. We also thank S. Hutchinson for carefully reading and commenting on some initial drafts of this manuscript.

## References

[1] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-D point sets," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-9, no. 5, pp. 698–700, 1987.

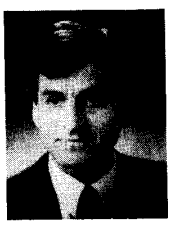
[2] B. Bhanu, "Representation and shape matching of 3-D objects,"

*IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-6, no. 3, May 1984.
[3] R. C. Bolles and P. Horaud, "3DPO: A three dimensional part orientation system," *Int. J. Robotics Res.*, vol. 5, no. 3, Fall 1986. pp. 3–26.
[4] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications.* New York: Elsevier, 1982.
[5] C. H. Chen and A. C. Kak, "A robot vision system for recognizing 3-D objects in low-order polynomial time," *IEEE Trans. Syst., Man Cybern.*, vol. 19, no. 6, pp. 1535–1563, Nov./Dec. 1989.
[6] C. H. Chen and A. C. Kak, "3D-POLY: A robot vision system for recognizing objects in occluded environments," Dept. Elec. Eng., Purdue Univ., West Lafayette, IN, Tech. Rep. 88-48, 1988.
[7] N. Deo, *Graph Theory with Applications to Engineering and Computer Science.* Englewood Cliffs, NJ: Prentice-Hall, 1974, pp. 314–321.
[8] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. Ass. Comput. Mach.*, vol. 19, no. 2, pp. 248–264, Apr. 1972.
[9] T. J. Fan, G. Medioni, and R. Nevatia, "Recognizing 3-D objects using surface descriptions," presented at the 2nd Int. Conf. Computer Vision, Dec. 5–8, 1988.
[10] O. D. Faugeras, "New steps toward a flexible 3-D vision system for robotics," in *Proc. 2nd Int. Symp. Robotics Res.*, 1985.
[11] O. D. Faugeras and M. Hebert, "A 3-D recognition and positioning algorithm using geometrical matching between primitive surfaces," in *Proc. 8th Int. Joint Conf. Artificial Intell.*, Aug. 1983, pp. 996–1002.
[12] O. D. Faugeras and M. Hebert, "The representation, recognition, and locating of 3-D objects," *Int. J. Robotics Res.*, vol. 5, no. 3, Fall 1986, pp. 27–52.
[13] H. N. Gabow, "An efficient implementation of Edmonds' algorithm for maximum matching on graphs," *J. Ass. Comput. Mach.*, vol. 23, no. 2, pp. 221–234, April 1976.
[14] Z. Galil, "Efficient algorithms for finding maximum matching in graphs," *ACM Comput. Surveys*, vol. 18, no. 1, March 1986.
[15] W. E. L. Grimson, "On the recognition of parameterized 2D objects," *Int. J. Computer Vision*, vol. 3, pp. 353–372, 1988.
[16] W. E. L. Grimson, "On the recognition of curved objects," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 11, no. 6, June 1989.
[17] W. E. L. Grimson and T. Lozano-Perez, "Model-based recognition and localization from sparse range or tactile data," *Int. J. Robotics Res.*, vol. 3, no. 3, Fall 1984.
[18] W. E. L. Grimson and T. Lozano-Perez, "Localizing overlapping parts by searching the interpretation tree," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-9, no. 4, July 1987.
[19] J. Han, R. A. Volz, and T. N. Mudge, "Range image segmentation and surface parameter extraction for 3-D object recognition of industrial parts," in *Proc. IEEE Int. Conf. on Robotics and Automation*, Mar. 31–Apr. 3, 1987, pp. 380–386.
[20] R. M. Haralick and G. L. Elliott, "Increasing tree search efficiency for constraint satisfaction problems," *Artificial Intell.*, vol. 14, pp. 263–313, 1980.
[21] R. M. Haralick and J. S. Kartus, "Arrangements, homomorphisms, and discrete relaxation," *IEEE Trans. Syst., Man Cybern.*, vol. SMC-8, no. 8, Aug. 1978.
[22] R. M. Haralick and L. G. Shapiro, "The consistent labeling problem: Part I," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-1, no. 2, May 1979.
[23] M. Hebert and T. Kanade, "The 3D-profile method for object recognition," presented at the IEEE Computer Society Conf. Computer Vision and Pattern Recognition, San Francisco, CA, June 19–23, 1985.
[24] J. E. Hopcroft and R. M. Karp, "A $n^{5/2}$ algorithm for maximum matching in bipartite graphs," *J. SIAM Comp.*, vol. 2, pp. 225–231, 1973.
[25] D. P. Huttenlocher and S. Ullman, "Object recognition using alignment," in *Proc. 1st Int. Conf. Computer Vision*, 1987.
[26] K. Ikeuchi, "Generating an interpretation tree from a CAD model for 3D-object recognition in bin-picking task," *Int. J. Computer Vision*, vol. 1, no. 2, pp. 145–165, 1987.
[27] A. Itai, M. Rodeh, and S. L. Tanimoto, "Some matching problems for bipartite graphs," *J. Ass. Comput. Mach.*, vol. 25, no. 4, pp. 517–525, Oct. 1978.
[28] A. K. Jain and R. Hoffman, "Evidence-based recognition of 3D objects," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 10, no. 6, 1988.
[29] T. Kim and K. Y. Chwa, "An $O(n \log n \log \log n)$ parallel maximum matching algorithm for bipartite graphs," *Inform. Process. Lett.*, vol. 24, pp. 15–17, 1987.
[30] L. Kitchen and A. Rosenfeld, "Discrete relaxation for matching relational structures," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-9, no. 12, Dec. 1979.
[31] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Res. Logist. Quart.*, vol. 2, pp. 83–97, 1955.
[32] Y. Lamdan and H. J. Wolfson, "Geometric hashing: A general and efficient model-based recognition scheme," in *Proc. 2nd Int. Conf. Computer Vision*, 1988.
[33] D. G. Lowe, "Three-dimensional object recognition from single two-dimensional images," *Artificial Intell.*, vol. 31, 1987.
[34] T. Lozano-Perez, J. L. Jones, E. Mazer, P. A. O'Donnel, and W. E. Grimson, "Handey: A robot system that recognizes, plans, and manipulates," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1987.
[35] D. W. Murray, "Model-based recognition using 3D shape alone," *Computer Vision, Graphics and Image Process.*, vol. 40, pp. 250–266, 1987.
[36] M. Oshima and Y. Shirai, "Object recognition using three-dimensional information," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-5, no. 4, July 1983.
[37] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity.* Englewood Cliffs, NJ: Prentice-Hall, 1982.
[38] A. A. Requicha, "Representations for rigid solids: Theory, methods, and systems," *Comput. Surveys*, vol. 12, no. 4, pp. 437–465, 1980.
[39] A. Rosenfeld, R. A. Hummel, and S. W. Zucker, "Scene labeling by relaxation operations," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-6, pp. 420–433, June 1976.
[40] L. G. Shapiro and R. M. Haralick, "Structural descriptions and inexact matching," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-3, no. 5, Sept. 1981.
[41] S. L. Tanimoto, "Analysis of biomedical images using maximal matching," in *Proc. IEEE Conf. Decision & Control*, Dec. 1–3, 1976, pp. 171–176.
[42] S. Umeyama, T. Kasvand, and M. Hospital, "Recognition and positioning of three-dimensional objects by combining matchings of primitive local patterns," *Computer Vision, Graphics, and Image Process.*, vol. 44, pp. 58–76, 1988.
[43] A. K. C. Wong and S. W. Lu, "Representation of 3-D objects by attributed hypergraphs for computer vision," in *Proc. 1983 Int. Conf. Syst., Man, Cybern.*, pp. 49–53.
[44] A. K. C. Wong and S. W. Lu, "Recognition and knowledge synthesis of 3-D object images," in *Proc. 1985 Computer Vision and Pattern Recognition*, pp. 162–166.
[45] A. K. C. Wong, S. W. Lu, and M. Rioux, "Recognition and shape synthesis of 3-D objects based on attributed hypergraphs," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 11, no. 3, Mar. 1989.



**Whoi-Yul Kim** received the B.S. degree in electronic engineering from Hanyang University, Seoul, Korea, in 1980. He received the M.S. degree from Pennsylvania State University, University Park, in 1983 and the Ph.D. degree from Purdue University, West Lafayette, IN, in 1989, both in electrical engineering.

Since 1989 he has been on the faculty of the Erik Jonsson School of Engineering and Computer Science, the University of Texas at Dallas. His current research interests include CAD model-based 2-D and 3-D recognition, computer graphics, artificial intelligence, factory automation, and computer integrated manufacturing.



**Avinash C. Kak** (M'71) is a Professor of Electrical Engineering at Purdue University, West Lafayette, IN. His research interests are in reasoning architectures for solving spatial problems, sensor-based robotics, and computer vision. He has coauthored the books *Digital Picture Processing* (Academic Press) and *Principles of Computerized Tomographic Imaging* (IEEE Press).