Image Interpretation and Object Recognition in Manufacturing

M. Marefat and R. L. Kashyap

A framework for intelligent interpretation of images in manufacturing applications is needed. Currently, most model-based vision programs are developed for a specific task in such a way that the knowledge of the task and the environment is implicitly coded into the system. It is difficult to modify the knowledge or extend the scope of such systems, and they require long development time.

In order to address the needs for modifiability, extensibility, and ease of tailoring for a particular environment the generic knowledge of the system is separated from the specific task knowledge. In the proposed framework, three types of knowledge are incorporated: the knowledge about objects, the knowledge about relations, and the knowledge about reasoning and problem solving. The relational knowledge, which includes both geometric relations, such as perpendicularity, and physical relations, such as on_top_of, is the major source of constraints imposed on the entities during the course of interpretation. We will show how constraints are propagated between the related entities.

To overcome the complexity of the task in the manufacturing domain, the objects are represented in three levels of primitive geometric entities, perceptual structures, and volumetric primitives. These representations can readily be obtained from the solid models of objects produced by CAD systems. The problem solving process consists of bottom-up and top-down components. It progresses by extracting primitive entities, forming the best structures represented by them, and using the relations between these structures to find the volumetric primitives. The best matches start a top-down reasoning process, which ultimately determines the primitive entities (edges and vertices) missing in the image, or finds the hypothesis about the object incorrect. A prototype system is being implemented which uses C for low level processing and LISP for reasoning and higher level tasks.

Introduction

Model-based vision systems deal with a practical and promising class of problems, that of understanding the semantic information in one or more images of a scene. These systems can be used for part identification, part loca-
tion, part inspection, and part manipulation tasks in manufacturing applications. Attempts to build such systems in different applications have proved that they must be able to deal with missing or extraneous features due to noise or incomplete information supplied by the low level processing units, partial occlusion, and the ambiguities due to the projection of the three dimensional scene to two dimensional images. To overcome these difficulties, domain knowledge has been used by the earlier vision systems to generate reasonable assumptions about the objects. In addition, the system must be organized such that its geometrically about the objects. In addition, the system must be designed to extract features, and also to recover three dimensional information from two-dimensional images.

The shortcoming of the model-based vision programs is that, currently, most model-based vision programs are developed for a specific task in such a way that the knowledge of the task and the environment is implicitly coded into the system. It is difficult to modify the knowledge or extend the scope of such systems, and they also require long development time. In addition, these systems also suffer from the following two problems: (1) the knowledge of the system must be manually tailored for each part, and (2) the vision system is not integrated with the engineering database used by other components of the automated manufacturing system.

In order to develop systems which are effective and efficiently adaptable for different tasks, knowledge-based vision systems must be able to represent generic objects and reason geometrically about the objects. In addition, the system must be organized such that its generic knowledge, such as the meaning of a perpendicular relation between two entities and the constraints implied by this relation, is separate from the specific task knowledge which may change. This aspect makes it possible to tailor or extend the system for another task with little effort. In this paper, we present our current work in developing and experimenting with a framework based on this approach for intelligent interpretation of images in manufacturing applications. This framework encompasses the issues surrounding the knowledge representation, reasoning and control, and maintenance of consistency.

Three types of knowledge are incorporated in the proposed framework, the knowledge about objects, the knowledge about relations, and the knowledge about reasoning and problem solving. The object knowledge includes the representations for the generic objects to be found in the environment, and specialized instances of these objects. The relational knowledge includes geometric relations, such as perpendicularity, and physical relations, such as the on_top_of relation. The knowledge of relations is one of the major sources of constraints imposed on the entities during the course of interpretation. We will propagate constraints between the related entities in each layer, and also up and down the layers of the hierarchical representation of objects. The problem solving and reasoning knowledge should be represented declaratively, so that it consists of modules which are easy to add to or modify. Hence, while we exploit the object oriented methodology to represent objects and relationships in a hierarchical manner, we use production rules to represent the geometrical and physical reasoning knowledge.

Image interpretation is difficult in the manufacturing domain, because of the inherent three-dimensional nature of the task. To overcome the complexity we represent the objects in three levels of primitive geometric entities (like edges and faces), perceptual structures (like perpendicular pairs of lines and different junction types), and volumetric primitives (like pockets and slots). The problem solving process consists of bottom-up and top-down components. It progresses by extracting primitive entities from the output of the lower level processing units, forming the best structures represented by them, and using the relations between them to find the volumetric primitives. The volumetric primitives are higher level and provide a more efficient indexing mechanism into the object database.

Once a match is proposed, the position and orientation of the object is determined by estimating the parameters of a transform relating the object's image coordinates to world coordinates using a least squares fit. The reasoning and selection process is guided by the constraints which are generated from: 1) the generic properties in the representation of objects, such as impossible junction combinations; 2) consistency of the view transformations derived from matching different groups of points between the image and the model; and 3) the specific properties of the object like perpendicularity between two planes. The best matches start a top down reasoning process, which ultimately determines the primitive entities (edges and vertices) missing in the image, or finds that the hypothesis about the object is incorrect.

The remainder of this paper is organized as follows. In the next section we briefly review some of the related works in both areas of knowledge-based image interpretation and CAD-based vision. In section three, we examine the manufacturing domain in more detail, and present some of the characteristics and peculiarities of the vision problems in this domain. The remaining sections describe the proposed framework. Section four introduces the knowledge representation in this system, and sections five and six deal with the issues of reasoning, control, and determining model to image transformations. Finally section seven takes a close look at an example for determining the identity, and the position and orientation of a part. The concluding remarks are presented in section eight.

Related Work

The related work in the area of image interpretation mostly concentrates on intelligent interpretation of aerial images [1]-[7]. ACRONYM [1] is one of the earlier systems built for this task. The objects are presented by generalized cylinders and subpart relationships using a hierarchy of frames. To perform matching between the image and the models the models are mapped into ribbons and ellipses which correspond to two-dimensional projections of the models. The reasoning in ACRONYM is basically of the algebraic form, performed by manipulating the inequalities derived from the tolerances on object models and the view point transformations. The MOSAIC system [2],[3] uses geometric reasoning to generate a description of the buildings in the scene from images taken from multiple view points. The object models are assumed to be rectangular prisms. The domain knowledge is implicitly coded into the MOSAIC system, therefore it is difficult to expand its knowledge or modify it to accommodate a new task. Walker et al. [17] are developing the 3D FORM system as an improvement. Their system uses frames to represent objects and geometric features. Completeness values are used by the system to choose the next match to be tried so that the most completed objects are matched first.

The related work in the area of CAD-based vision [8]-[15] has basically concentrated on part recognition. Hoffman et al. [8] report two experiments, one based on intensity images and another based on synthetic range images, for automatic identification and location of objects using their characteristic features (edge contours). The observed characteristic features are used for identification, and the comparison of the surface patches of the characteristic features and the faces from the CAD boundary model, of the object is used for pose estimation. However, "it is necessary to specify the kind of information which make up characteristic features. The low level features...are manually specified" [8]. Park and Mitchell [11] have developed a system for CAD-based visual inspection. Their system assumes that the objects are designed with a feature-based approach. A scoring mechanism is used to choose the image features (edges and circular arcs) to look for. The geometric
reasoning in the system is limited to rules for determining visible entities in the interaction between the set of machining features which are assumed to have been used in the design of the part. Hansen and Henderson [10], present methods for automatic generation of recognition strategies based on CAD models. The Alpha-I solid modeling package is used in their experiments. A set of filters is sequentially applied to the model of the object to quantify properties of its features (edges) in terms of robustness, completeness, cost, and uniqueness. The results are then used to construct recognition schemes called strategy trees.

It is also interesting to note that a neural network approach has been developed for three-dimensional object recognition [14].

**Domain**

The task of image interpretation in the manufacturing domain is difficult. Unlike the systems built for the detection of buildings in aerial images, this domain is not limited to rectangular buildings. Also, in systems proposed for detection of buildings in aerial images, the system is, typically, concerned with accumulating edges into groups which represent good roof hypotheses (that is groups which don’t overlap and are, for instance, supported by shadow edges), but the objects may be viewable from many different perspectives in the manufacturing domain. Therefore, there is a greater need for three-dimensional reasoning.

Identification, verification, visual inspection, and part location are representative of the problems which must be solved in different manufacturing applications. The identification problem, like that of finding objects in a robot’s work space, is concerned with determining the identity and approximate position and orientation of the part(s) in a scene. In verification problems, like object monitoring or checking a part’s presence in a feeder, the identity of the part is known but one needs to make measurements to verify the part. The visual inspection problem also assumes that the identity of the object is known, but dimensional measurements must be taken to make sure that features like holes and slots of a part are machined correctly along with their dimensions. In part location problems, like picking an object from a part’s bin, one must determine the exact position and orientation of a (possibly partially occluded) part. Whatever the problem, the vision system must be able to handle imperfections, such as an incomplete set of detected features and errors in feature locations.

In order for the vision system to achieve any of the above tasks with reasonable efficiency it must intelligently exploit the natural constraints embedded in the task in its reasoning. Some of the important constraints implied by problems in this domain are the following:

- Flat support constraints are common in identification problems. Typically, it is assumed that objects lie on a table in a stable configuration. Therefore, their vertical position is constrained, and for each stable configuration (determined by face(s) in contact with the table) only rotations around one axis, the vector perpendicular to the table, is considered.
- Known position constraints are common in part location problems. In such tasks as bin picking, the position of the object is constrained to be within the limits of the bin, but the part may be oriented in any manner about the three possible rotation vectors.
- Known position and orientation constraints are encountered in verification and visual inspection. In these tasks parts approximate position and orientation are known, so the allowed range of variation is limited.

In order to attain an integrated environment, the aim of many projects has been to use solid models, produced during the design phase in the CAD database, for vision purposes. There are several ways to describe the solid models of objects, including spatial enumeration, primitive instancing, sweep representations, cell decomposition, Constructive Solid Geometry (CSG), and boundary representations [15]. However, most solid modeling systems use either CSG or boundary representations. In this research we use boundary representations as the solid model of objects produced by the CAD system. The boundary representations describe the object in terms of the low level topological entities (faces, edges, and vertices) from its boundary. However, indexing into the databases of objects is more efficient if higher level primitives, like volumetric or functional primitives, are used rather than using only lower level primitives, like edges. Therefore, we have developed methods [16] to automatically extract a set of higher level primitives, like pockets and slots, of a part from its boundary model, and supplement the boundary models with higher level descriptions in terms of these semantic primitives and relations between them.

**Knowledge Representation**

There are three major parts to the representation scheme used in our framework: 1) knowledge about objects, i.e., features and data which describe the objects; 2) knowledge about relations between objects such as perpendicularity and on top of relations; and 3) problem solving knowledge which uses production rules to make inferences over objects and relations.

**Model Knowledge**

The information used to represent the objects is stored using frames. Each frame contains slots in which descriptions of the object are stored. These parameters may simply be assigned from the information collected from the images or can be computed in the frame

---

Fig. 1. Schematic diagram for model-based interpretation.
itself. For example, we may collect information such as starting and ending points of lines and the frame may also have a slot in which the lengths of the lines are computed. These frames are arranged in a hierarchy with the top level containing overall information regarding the object, and the lowest level frames containing detailed information about particular facets of the object such as edges and junctions. Currently our system can represent objects with planar and cylindrical surfaces. The frames for the overall object are as follows for our particular system:

- **object**
  - outer faces: list of outer faces
  - depressions: list of depressions

where an outer face is a face which is not a part of the interior of a depression. These two subcategories are then broken down into their respective lower level categories which contain more detailed information regarding specific features. The frame for an outer face contains information about the edges in the boundary of the face and the normal vector to its plane:

- **outer face**
  - edges: list of edges (Each edge is then represented by its own frame in which the length and direction are computed and the position is represented.)
  - normal: unit normal vector to the face (The normal slot has a method associated with it. Therefore, it can be set by the information collected in the reasoning process or computed from the edges in the list of edges.)
  - openings: depressions whose openings lie on the face;
  - and the frame for a single depression of the object looks like:
    - **depression**
      - type (hole, pocket, slot, blind-slot, step, or blind-step)
      - depth
      - position image edges comprising opening (This is instantiated when a particular image is inspected for primitive openings.)
      - opening face.

The subcategory for an edge also contains a slot for categorizing that edge’s intersections with other lines. There are only four types of intersections: the L, the Fork, the T, and the Arrow in a trihedral (three-faced vertex) world when there are no shadows or cracks and the objects are in a general position (no junctions change type with small eye movement) [17]. These intersection types indicate the shape of the object at the given corner. A partial example of an object frame is given in Fig. 2. Note that these representations are for our particular system. They may be changed or new ones added to them as the objects represented in the database of the system change.

To aid in filling the slots of each frame, we utilize a data structure called a view graph. These graphs use the data generated by the solid modeler in the form of boundary representation to predict the features visible in images from a particular view. The change in appearance of objects in different scenes can be divided into two sets of changes, linear and nonlinear. Nonlinear shape changes are those in which the observable faces of the object are different from one view to another. Linear shape changes are those in which the same faces of the object are visible, but the set of observable features is different because of the change in view. Initially, view graphs are used to determine the approximate pose and orientation of an object as seen in an image once that model has been selected from the database. View graphs are actually hypergraphs that represent nonlinear and linear shape changes in terms of the observed features of the object. Each hypermode represents a collection of connected linear shape changes: the set of observed faces is the same for each hypermode. A hyperarch connects two hypermodes if the region of space from which one hypermode is seen is adjacent to (intersects along the same surface) the region of space from which the faces in the other hypermode are seen. Each hypermode itself consists of an underlying graph representing the space of linear shape changes. In this underlying graph, two nodes are connected if the portions of space from which these views are seen are adjacent. The nodes of this underlying graph contain the set of observable features such as junction types, circuit codes, and observable relations like parallelism, collinearity between line segments, and connectivity between circuits from that node’s particular view.

Constraints Knowledge

To determine whether the predicted (and hypothetically observable) relations do exist, we have a section in the representation devoted to capturing the relations between features. Object oriented methodology is used to capture the constraint knowledge. Different constraint types are represented by different classes. Individual constraints are instances of these classes. Each constraint is therefore represented by a frame which has slots for the participants of the constraint, and methods which can be invoked by sending messages to the constraint. For example, we implement the perpendicular edge constraints as a frame with

---

**Fig. 2. Partial instantiation of an object frame for an image.** If viewed from a different direction, the frame would differ according the faces viewable. Object: o1; outer faces: s1, s2, s3, s4, s5; depressions: sl, s2, e1, e2; edges: e1, e2, e3, e4, e5, e6, e7, e8; normal to face: z. Edge: e1; starting point: (15, 97); ending point: (15, 142); slope: infinity; intersections: L: e8. ARROW: (12, 10); Depressions: slot1, slot2; slot1; type: slot; depth: 1.0; position: (90, 2.0, 3.0); edges of opening: e10, e11, e12; opening surface: s2.
Methods are functions which are invoked when an appropriate message is sent to the constraint. Constraint propagation is achieved by message passing in this approach. For example, there are methods in a constraint which determine whether the constraint is satisfied, and there are also methods which determine which candidates from a pool (like line segments) can be used to satisfy a given constraint. Messages can be sent to a constraint to ask it to provide a list of its participants, and also messages can be sent to each participant (such as a line segment) to ask it to provide a list of all constraints it participates in.

In case of perpendicularity, to check that the relation holds, we can also send a message to one of the edges with the second edge as argument. The method associated with the message is invoked, which returns whether the relation holds. If it does, a new instance of the constraint class is instantiated. The instances are stored in appropriate lists.

The relational knowledge is divided into two categories, one for geometric relations such as perpendicularity, parallelism, tangency, etc. between low level features in the image, and the other for physical relations between high level features in the image. This second subgroup gives us information as to where one object or depression in the image lies in relation to another. An example of this might be "slot1 lies to the left of step2."

The geometric relation knowledge can be easily implemented by LISP functions attached to methods. For example, a method to check the perpendicularity of two edges could look like the following pseudo-LISP function:

(parallel (edge1 edge2)
 (and (intersect edge1 edge2)
 (closero (angle_between edge1 edge2) 90)))

where closero is a function which determines within a given amount of error whether the angle of intersection was 90°. The function itself determines whether the two lines given to it intersect and if they do, whether they do so at a right angle. This helps to constrain the search space when matching image lines to model lines in three different situations. The first situation is if we are searching for two perpendicular lines and neither of them has been found. The approximate point of intersection is known from the model to image transformation so the system needs only to look at lines intersecting at this point with the characteristic of perpendicularity. A second scenario is when one line of a perpendicular intersection has already been found by the system and the other must be located. In this case, only those lines which are perpendicular to the given line must be checked for the correct intersection point. If both lines in the intersection have been found, the relational knowledge is still useful in verifying the condition of perpendicularity for top-down verification. The same strategy has been implemented to determine whether two lines are parallel using their slopes and distances.

Physical relations are somewhat more difficult to represent due to the fact that these relations deal with a higher level of representation and therefore cannot be written in the same low level process-compare style. These functions must take into account more information and are thus more complex. For example, we may have determined from our view graph node that a particular depression, say slot2, must lie to the right of another depression, say pocket1. To determine if this is indeed the case we would invoke our relational knowledge representation function for "to the right of" as follows:

(right_of (depression1 depression2)
 (setq max1 (get_max_x (get_edges depression1)))
 (setq min2 (get_min_x (get_edges depression2)))
 (greaterp min2 max1)).

This would return whether the leftmost point of depression2 had a greater x value than the rightmost point of depression1, thus determining whether the entire opening of depression2 was to the right of the entire opening of depression1.

The same type of relations can be determined for entire objects if there is more than one object in a single image. To determine these relations, we use the list of outer faces of the objects. For example, if we wanted to know if object 01 lay on top of object 02, we would invoke the "on top of" relation for objects:

(on_top_of (object1 object2)
 (and (opp_normal (get_bottom object1) (get_top object2))
 (greaterp (get_y (get_top object1)) (get_y (get_top object2))))).

This finds whether the normals of the top plane of object 02 and the bottom plane of object 01 are in opposite directions (within a constraint) and whether the bottom plane of object 01 is higher than the top plane of object 02. By applying constraints using this relational knowledge, we may eliminate some of the candidates for the general view which do not fit the relations. This prunes our search space and helps us to find the appropriate node in the view graph more quickly.

**Problem Solving Knowledge**

After we have determined the approximate position of the object, the view graphs also allow us to determine which features should be observable from the particular view. Using this information we are better able to fill in the slot representing the list of depressions in the frame giving the overall view of the object. To accomplish this we attach a set of descriptions and instructions to each node of the view graph. The descriptions tell us which features are most likely observable from the given view, thereby forming predictions of what should be seen in the image in terms of features and relations. The instructions attached to the node tell us how to interpret the scene. They are in the form of if-then production rules and specify which relations should be sought and how their measurements should be used to gain numerical three-dimensional understanding of the scene.

The instructions just mentioned are a part of our third division of representation, problem solving knowledge. This part of representation gives us the rules for what types of things to look for when processing image and model information. An instruction in the view graph might specify, for instance, that if a line is found beginning approximately at point (x,y) with a slope about m in the lower left, to look for a line parallel to it at a certain distance in a given direction. This is an example of how view graphs can provide instructions to lower level processing units to look for those features which were not found in the original processing. Due to the inaccuracies of edge detection resulting from poor lighting, too high a threshold, or other factors, it is very rare that all features and relations between the image and the node of the view graph are found in the original processing. This problem solving knowledge attached to the view graphs allows, for example in the case of an edge detector, to look in a smaller region adjusting its threshold locally to find a segment or junction which was omitted initially. We can, therefore, more completely fill in information regarding specific instances of primitives which was initially left out. In this way, we obtain a more complete description so that we may perform more accurate and efficient reasoning.

**Reasoning and Control**

The geometric reasoning phase of our framework begins with a bottom-up approach. We start with an edge detection step to gain low level information about the object in the image. From this, we gain a list of lines seen in the image. We then begin some intermediate processing of this data. This processing includes finding the corners of the lines and their junctions with other lines. The junctions are further specified by how many lines are in-
volved in the intersection and the angles of intersection. This information forms a rich base from which we may build our higher level processing.

When performing image to model matching, we must first determine the object being viewed and the pose of that object. To determine this general information, we trace edges in the image searching for loops and label the edges looking for patterns indicating the shape of the object. For example, we may begin at a corner of the object in the image and trace around an entire face of the object labeling each corner of the tracing as either concave or convex depending on whether the corner turns less than or greater than 180°. This is done for each of the faces viewed in the image, thus filling in the information in the outer surface frames of the global object information frame. Predictions of which volumetric primitives exist in the image are made by computing the angles of intersection of joined lines found when tracing faces of the image. Each intersection is marked as either concave or convex (as above) and patterns which indicate the nature of the primitive are searched for. Since we do not place any specifications on the measures of the angles used to indicate primitives, the system is not restricted in to only objects with perpendicularly intersecting faces. The system can also detect primitives which interact. An example of this is shown in Fig. 3. The pattern for a slot is 1001 while the pattern for a step is 101. In the object face shown, the pattern of the entire tracing is 110010111. Both the patterns 1001 and 101 exist in the sequence but they overlap. This indicates that the slot and the step in the object are not completely separate entities but instead interact. This fact is also used when matching an image to its model.

When all of this information is accumulated, we may perform a search of the view graphs in the database to determine which object is being seen in the image. The exact view graph of the object is chosen from the group of graphs representing that object by performing a more precise search on this smaller group of graphs. This condensed search is done by matching particular depressions seen on faces in the image with those modeled in the view graphs. For this search we must use information regarding the positioning of depressions seen with respect to one another. (These depressions are identified by recognizing patterns in the edge labelings done above.) For example, if we see an opening to a prismatic hole and an opening to a slot, we check our relational knowledge representation in the view graph to see what the relation of the positions of the depressions should be (e.g., "pocket to the left of slot"). We continue comparing instances and positionings of particular types of features (without regard to dimensions at this point) until we have narrowed our search space to one particular view graph which we take to be the general view from which we are looking at the object.

With the general positioning now established, we must determine the specific pose of the object as seen in the image. This is accomplished by matching points in the image with points in the boundary representation. Since we know the approximate pose of the object in the image, we are able to determine the relation of several points in the image (e.g., the outer corners) with their respective points in the model. From this information, we can perform a least squares operation to obtain an accurate transformation from the model to the image. With accurate view graph information we are also able to perform a successful two-dimensional to three-dimensional matching since we know which features are visible in the image and which are hidden. In most cases this matching is extremely difficult if not impossible since a two dimensional point normally maps to a line when transformed to three dimensions. It should also be noted here that there is a tradeoff between computing time and accuracy when using a least squares transformation. Matching more corners produces a more accurate transformation matrix but at the cost of computation time.

Now that we have view direction and positioning established, we are ready to begin the matching phase of our framework. To begin, the image is inspected for probable openings of depressions as described above. We are now able to determine the exact positioning of these openings by using our image to model transformation. For each feature found in the image, we compare the information to the frame corresponding to the feature, e.g., position, dimensions, direction. This information is then grouped and compared to the highest level frame for the overall representation of the object. The chances are very great that this data will not match exactly due to inaccuracies in the collection of information by the line finding algorithm. These conflicts in information trigger a top-down verification process from model to image. The instructions attached to the view graph node are then applied to the image data to verify whether the given model information is consistent with what exists in the image. Each one of these matchings helps to confirm or refute the validity of the overall image/model matching. Once initial verification is done, we perform a check using a new transformation to make certain that our initial transformation was accurate.

If we conclude that the object in the image does not match the model, there are two possible sources for the problem. The first is that the system may have chosen the incorrect object from the database. For this possibility, we keep track of the models in the order that they were eliminated from the search. If the object were judged incompatible with the selected model, we may check the next most likely model for a match. If this does not match, we could check the next model, and so on until either a match is found or the models become so incompatible that we decide that the image match no model in the database. The latter case is our second possibility. This judgment could have different implications depending on the application of the system. In an automatic inspection environment, this decision could cause us to examine the manufacturing machinery to make sure that it is operating correctly. In an assembly situation, the given object could be placed in an "unusable" bin. In any case, action should be taken when a defective part is found.

Thus, our reasoning scheme includes both bottom-up and top-down approaches for determining model/image matching. The search area in the images is gradually refined to allow efficiency in both searching the database and error handling.

Position and Orientation

In order for a system to perform image interpretation, there must be a method for mapping model points to the image so that it can be determined whether a model point is observable in the image. This is commonly done by matching model points to their respective image points and computing the perspective transformation. The transformation takes the form of a 4 by 3 matrix \( T \) such that when it is premultiplied by a model point in the form \((x_o, y_o, z_o, 1)\), it yields a vector of the form \((u, v, 1)\), where \((x_o, y_o, z_o)\) is the model point and \((u/v, 1)\) is the image point matched to the model point. To find the elements \( T \) of the matrix \( T \), we rewrite the equa-
tion \((x,y,z)\) for a matched points in the form:

\[
\begin{bmatrix}
x_m & y_m & 1 & 0 & 0 & 0 & 0 & 0 & -x_m & -x_m & -x_m & -x_m \\
0 & 0 & 0 & x_m & y_m & 1 & -y_m & -y_m & -y_m & -y_m \\
x_m & y_m & 1 & 0 & 0 & 0 & 0 & 0 & -x_m & -x_m & -x_m & -x_m \\
0 & 0 & 0 & x_m & y_m & 1 & -y_m & -y_m & -y_m & -y_m
\end{bmatrix}
\begin{bmatrix}
T_{11} \\
T_{12} \\
T_{13} \\
T_{14}
\end{bmatrix}
= \begin{bmatrix}
x_m \\
y_m \\
x_m \\
y_m
\end{bmatrix}
\]

with \(T_{14}\) set to 1, as discussed in [18].

The above system is in the form \(A \mathbf{x} = \mathbf{b}\) where \(A\) is an \(m\times n\) matrix, \(\mathbf{x}\) is an \(n\times 1\) vector representing the transformation parameters in a vector form, and the right hand side of the equation is the vector containing the coordinates of the image points used for matching. Each image point contributes two equations to this linear system (one for the \(x\) coordinate and one for the \(y\) coordinate). Although this allows us to find an exact transformation by matching 5 1/2 points, it does not allow for any of the matched points to be slightly in error. If a point in the image has been detected incorrectly, then the entire transformation matrix will be off. This leads to poor results and incorrect or unreliable conclusions.

To avoid this problem, we employ a least squares method for determining the transformation parameters using more than 5 1/2 points to match. If the correct transformation parameters are represented by \(\mathbf{t}\), then the least squares solution is \(\mathbf{t} = (A' A)^{-1} A' \mathbf{b}\), where \(A'\) is the transpose of \(A\). Note that \(A' A\) is a square matrix and the new system \(A' A \mathbf{t} = A' \mathbf{b}\) can be solved if \(A' A\) is nonsingular.

The new system yields the least squares solution for the original system. Such an approach allows us to use as many points as desired to solve for the transformation. Greater number of points yield greater accuracy.

**Example**

This section shows an example of how the system uses bottom-up and top-down reasoning to match a particular image containing one object to its model in the database. Attention is given to completing the information left out by the line finding algorithm and disregarding extraneous lines due to noise in the image. We first require photographs of a scene containing one or more objects. The bottom-up phase of processing is then invoked to make the initial matching into the view graphs of the database. In this particular example, the scene contains one object. The images are shown in Fig. 4. In the first step, the edge segments in the image are determined. We use an edge detection algorithm by Nevatia and Babu [19] which provides detailed information on the lines found in the image. An image of the output of the edge detector can be seen in Fig. 5. When the short edges are eliminated and close edges are merged, the resulting image looks much like the outline image in Fig. 6. The information also comes in the form of a list of lines and their intersections with other lines. Lines which come close to intersecting (within five pixels) but do not touch are also found and marked accordingly. A line is a sequence of edges that follow each other so as to connect a start point and an end point by a path. Corners along individual lines are found by performing a linear approximation by joining the endpoints of the line. The point of maximum error is labeled a corner and is used to form two subsegments, one to each of the original endpoints. The process is repeated recursively until the error is within an acceptable bound. Each line is now characterized by start and end points, pixel length, and a list of corners along the line.

The next step is to create the initial entity frames for the lowest level entities found in the image from the data accumulated. We accomplish this by first creating the frames for edges and outer surfaces, then searching these surfaces for openings of depresions by looking for convexity/concavity patterns in the edges of the surface. The surface frames are created by beginning at a point on the image and attempting to trace a loop back to that point. Heuristic rules are used to assure that all edges traced in the loop lie on the same surface. These heuristics are simple rules which use the types of junction (L, T, fork or corner) at the ends of a segment, and the label of the edge to decide which leg of a junction at the end of a segment should (preferably) be followed. For example, if we are traversing the shaft of a \(T\) in the direction into the \(T\) junction, we do not continue traversing, since \(T\) junctions usually arise as a result of one face hiding part of another face. These heuristics are based on the trilateral world assumption and are implemented as a collection of if-then rules. If at any time the loop tracing algorithm becomes unable to continue due to a missing line, heuristics are also employed to hypothesize the most likely missing line so that the loop may be completed.

An example of the tracing of a loop can be found in Fig. 6 (dashed line segments are hypothetically considered by the system).

Note that edge \(e_1\) is a hypothetical line created by the system to complete the loop beginning at point \(p_1\) and proceeding in a counterclockwise direction. As the loops are traced, we make note of the angle of intersection at each corner. If a corner turns less than 180°, it is labeled as concave (denoted \(I\)). Otherwise, it is labeled as convex (0). The sequential patterns of these labelings are recorded to enable recognition of primitive openings. The pattern for surface \(s1\) is \(11100111\), and for surface \(s2\)
Hypotheses are generated by searching each string for specific patterns. For example, a pattern of 1001 usually indicates the opening of a slot. Thus, we may mark the surface containing this pattern as having an opening of a slot. This type of information is used to determine the exact node of the view graph. The information in the frame hierarchy is continuously updated. Our example part contains three slots, two of whose openings lie on the same face. Note that surface s6 is the result of an erroneous hypothesis by the system. The system added edge e3 in an attempt to complete a loop beginning at point p3. This error was caused by the extraneous edge e2 along with the partial occlusion by surface s3. Surfaces with hypothesized edges are marked as such so that the top-down procedure can either verify or reject each hypothesis by checking the added information against that in the model.

The next step of the process is to determine which object from the database is being viewed and from approximately what position. To accomplish this, we must search our database for the most probable objects and perform a more detailed search on those objects. To aid in this process, we have our database organized with regard to the volumetric primitives in each object. Thus, since we have found only slots in our image, we will search only those objects containing slots. The database and organization of objects by the primitives in their boundaries appears in Fig. 7. The search proceeds by examining each object and comparing the volumetric primitives of the object with those hypothesized in the image. Relations are also compared using the gathered knowledge. For example, the part in consideration contains three slots. There are two objects in the database (refer to Fig. 7) which contain three slots (parts 3 and 4). However, in part 4 the openings of these slots all lie on different planes, whereas in part 3 two of the slots lie on the same surface. Thus, part 3 is more consistent with the patterns found (Fig. 7) while tracing the outer edge loops and is accordingly assigned a higher probability of being the object in the image. Each object is assigned a probability that it is the object in the image reflecting how much the data in the image matched the object data. The object with the highest probability is chosen initially to be the object seen. In our example, the object with three slots (object 3, Fig. 7), two on one face and one on another, is given probability 0.875, the object with four slots (object 2, Fig. 7) 0.80, and the object with two slots (object 1, Fig. 7) 0.70.

In assigning probabilities, we have used the following methodology: the number assigned to belief in existence of an entity is a normalized combination of the existence of the primitives comprising the entity and the satisfaction of the expected constraints between these primitives. In evaluating existence of primitives such measures as the observed fraction of a line to its total length are used. In evaluating satisfaction of constraints, a predetermined margin of error is selected, and if the constraint is satisfied within that margin, it is assumed to be satisfied. In assigning probabilities, we also take care that the assigned numbers are consistent, i.e., the sum of the comprising probabilities add up to one, and no hypothesis score a negative probability assignment.

Once the object has been selected, the pose of the object in the image must be established. We accomplish this by examining which faces are showing and their relations with each other. In the image, the left face has one slot opening and the right face two slot openings. We find the view with matching information using probabilities in much the same way as we determined the object, from the nonlinear view information in the viewgraph of the object. There are four such views for this object, assuming that the object is lying flat on a horizontal surface and the camera is not perpendicular to any face.

Determining an exact transformation from model to image and back is the next task. As was mentioned in the reasoning section, we perform a least squares transformation to establish the relationship from model to image. In this example, we used seven noncoplanar points to determine the relationship, though five and a half would have sufficed. The wireframe of the model obtained from the transformation is presented in Fig. 8. As is clear by comparison with the image of Fig. 4, the transformation is accurate and reliable. The opening of slot 1 matches approximately to its hypothesized value, but as can be seen from Fig. 9, its length cannot be measured due to a missing edge. This triggers a top-down verification from model to image. The point at the end of the hypothesized slot is mapped to the image and the missing edge is hypothesized filled in. This is consistent with the data representing slot 3 which can now also be completed and measured in the same way. Note that the system is able to deal with interactions between primitives by simply adding in lines which do not exist due to interactions. An example of hypothesized information which is not consistent can be seen in Fig. 9(b) in this case the hypothesized edge added to complete slot 1 crosses the edge added to complete slot 3. Also, edge e3 of Fig. 6 is also found to
be inconsistent since its intersection point with surface s2 matches no point in the model. Edge e1, on the other hand, does match to a point in the model and is hypothesized to be an accurate addition. To confirm these hypotheses, the system computes a new transformation using different matching points in the model and image and performs the verification step again. In addition to checking hypotheses on added lines, this step also confirms the choice of the model itself. If our hypothesized object matches the image, this new transformation should give us results similar to the first. Otherwise, the system may have chosen an incorrect model originally. This is very useful when there are objects in the database with similar features but different sizes.

Since all of the information now matches (reasonably) with the object, it is ascertained that the object match is correct and that all of the measurements are correct. If the procedure had returned an overall low rating for matching, we would have used our second choice for the object (the model with four slots) and performed the same operations. The process would have been followed until either a match was found or it was determined that the image matched no model in the database.

Concluding Remarks

We have discussed issues that are important for knowledge-based interpretation systems and their desired characteristics in the manufacturing domain. We also presented a system that is being implemented based on these concepts. This system will become a part of a larger system that we are developing which integrates part design, process planning, and CAD-based vision [20],[21]. In our scene understanding system, the model knowledge is represented in a hierarchical manner in three levels of primitive geometric entities, perceptual structures, and volumetric and functional primitives. The knowledge of the relations is used to form constraints which are used for reasoning in the interpretation process. The interpretation process consists of the bottom-up and the top-down components. During this process, constraints are used to instantiate entities in different levels as well as verify hypotheses. This framework addresses the needs for extensibility and modifiability by separating the generic problem solving, constraint, and representation knowledge from specific model and task knowledge.

Efficiency is an important issue in discussion of image interpretation systems. We have tackled efficiency by incorporating two major strategies. First, we employ a hierarchical representation of knowledge about objects. This not only allows us to index into the model database with higher level volumetric and functional primitives, but also allows us to look for lower level primitives in the image, and then using proper constraints imposed on these entities, to hypothesize the higher level primitives observed in the image. This way we neither index into the database with low level primitives, nor search for higher level primitives in the image, solving both the efficiency and the difficulty of extraction of high level primitives. Second, we employ layered viewgraphs. We do not use a purely viewer centered representation of objects (in which the appearance of object from all different viewing directions is stored) for matching. We use our object centered representation for indexing. When a promising candidate is chosen we use the viewgraphs to determine pose and orientation and to verify. The lower layer of viewgraphs containing linear shape changes is only used for verification when a particular candidate is selected. Otherwise, using a flat viewgraph (consisting of low level line and vertex primitives) for both matching and verification would be very inefficient.

We have identified several areas of future research to be investigated further. First, constraint propagation must be studied further, so that the constraints can be structured better, and to allow the reasoning scheme use the effect of constraints in a more global manner. Second, we intend to incorporate better methods for algebraic reasoning, so that geometric and algebraic methods play complementary roles. Finally, an immediate extension of the presented work is to include objects which contain other quadric surfaces besides cylindrical surfaces in their descriptions.

References

three-dimensional objects for vision" in AI Mag., vol. 9, pp. 47-58, Summer 1988.


M. Marefat received the B.S. degree (Magna Cum Laude) with a double major in electrical engineering and mathematical sciences from Rice University, Houston, TX, in 1986, and the M.S. degree in electrical engineering from Purdue University, W. Lafayette, IN, in 1988. He is associated with the Purdue Engineering Research Center for Intelligent Manufacturing Systems and the school of electrical engineering, Purdue University, where he is currently pursuing the doctoral degree in electrical engineering. His research interests include geometric modeling, knowledge-based systems, machine perception, CAD/CAM, computer graphics, automation, and robotics. Mr. Marefat is a member of Tau Beta Pi, Eta Kappa Nu, and the Association for Computing Machinery.

R. L. Kashyap is currently a Professor of Electrical Engineering and Associate Director of the Engineering Research Center on Intelligent Manufacturing Systems at Purdue University, West Lafayette, IN. He is the recipient of many honors, including the 1990 King Sun Fu Award given by the International Association for Pattern Recognition for fundamental contributions to pattern recognition and computer vision, the fellowship of IEEE, the fellowship of Institute of Electrical and Telecommunication Engineers (India), and the best research paper award at the National Electronics Conference (1967). He is an area editor for the journals CVGIP: Graphical Models and Image Processing and The Journal of Intelligent and Robotic Systems. He has been guest co-editor of IEEE Transactions on Software Engineering (1988), IEEE Computer Magazine (1989), and the IEEE Transactions on Systems, Man and Cybernetics (1991). He has directed over 25 Ph.D. dissertations at Purdue. He has authored one book and over three hundred publications including ninety archival journal papers in areas such as pattern recognition and image processing, signal processing and system identification, random field models, intelligent data bases and intelligent manufacturing systems. He received his Ph.D. degree from Harvard in 1966 and has been at Purdue ever since. He has held visiting professor positions at Harvard University, the University of California at Berkeley, and the Indian Institute of Technology, Kanpur.