CAGD Based 3-D Visual Recognition

**Tom Henderson, Chuck Hansen, Ashok Samal,
C.C. Ho and Bir Bhanu**

Department of Computer Science
The University of Utah
Salt Lake City, Utah 84112

UUCS-85-115

2 December 1985

Abstract

A coherent automated manufacturing system needs to include CAD/CAM, computer vision, and object manipulation. Currently, most systems which support CAD/CAM do not provide for vision or manipulation and similarly, vision and manipulation systems incorporate no explicit relation to CAD/CAM models. CAD/CAM systems have emerged which allow the designer to conceive and model an object and automatically manufacture the object to the prescribed specifications. If recognition or manipulation is to be performed, existing vision systems rely on models generated in an *ad hoc* manner for the vision or recognition process. Although both Vision and CAD/CAM systems rely on models of the objects involved, different modeling schemes are used in each case. A more unified system will allow vision models to be generated from the CAD database. We are implementing a framework in which objects are designed using an existing CAGD system and recognition strategies based on these design models are used for visual recognition and manipulation. An example of its application is given.

This work was supported in part by NSF Grants ECS-8307483, MCS-82-21750, DCR-8506393 and DMC-8502115. Chuck Hansen is an ARO Fellow.

1. Introduction

Computer vision has been an active research area for over 20 years. In the early days, emphasis was on low level processing such as intensity and signal processing to perform edge detection [1, 19]. Systems were constructed which only operated in very constrained environments or for very specific tasks [2, 16, 20]. It was quickly recognized that higher level concepts of *image understanding* were needed to successfully perform computer vision. More recently, models of objects and knowledge of the working environment have provided the basis for driving vision systems. This is known as model based vision. The pursuit of the fully automated assembly environment has fueled interest in model based computer vision and object manipulation.

The problem we are interested in solving is model based visual recognition and manipulation of objects in the automation environment. This involves building a 3-D model of the object, matching the sensed environment with the known world and locating objects. Not until the desired object is located and its orientation is known can a robot gripper or hand manipulate it.

Our goal is to develop a system which will work in the environment of the automated assembly process. This is not intended to provide a general model for the human visual process but rather a solution to the problem of visual recognition and manipulation in a well-known domain. The constraint we are imposing is one which limits the necessity of modeling the entire world. Rather, the known world to us is that of the automated environment in which this system is intended to operate.

Simply stated, our approach is to provide an integrated environment in which the CAGD model can be used to generate appropriate recognition and manipulation strategies. A major aspect of this work is the successful development of a prototype system combining design, vision analysis and manipulation. In this paper, we describe:

1. the design of an object using the Alpha_1 CAGD system,
2. the analysis of 3-D range data to find the object,
3. the recognition of the object using a new multi-constraint discrete relaxation matching algorithm.

2. Object Design

The current modeling environment is the Alpha_1 Computer Aided Geometric Design (CAGD) System. It models the geometry of solid objects by representing their boundaries as discrete B-splines. It allows the construction of simple objects into a more complex object using set operations. It supports several modeling paradigms, including direct manipulation of the B-spline surface, creation and combination of primitive shapes, and high-level operators such as bend, twist, warp and sweep. By using set operations on sculptured surfaces, the modeling task becomes simpler and more complete than with other design systems.

The guidelines which are followed in using Alpha_1 are:

1. Analyze the object. Usually a complex object can be divided into simpler parts which can be more easily designed. A rule of thumb is "Use the same procedure that people use to

create the object." Once the procedure is decided, one can concentrate on each subpart.

2. Measure parameters. Make a precise measurement of parameters. If the design data is available, it is best to use it.
3. When the surface patches of each part are designed, make sure that they have the correct orientation and set the adjacency information correctly. Otherwise, an object which is not valid may be created.
4. Put all subparts in the correct position and orientation. Then use the combiner to perform the appropriate set operations.

Now we show the design of the object shown in Figure 1. The object is designed in a sequence of steps starting with the main plate, then adding indentations and all the holes. To obtain these, we design curves using B-splines first and then various high-level operators for surface construction; e.g., revolving a curve about an axis, extruding a curve, and filling in the surface between curves. Threads in the holes are designed by filling two surfaces between two twisted curves. The final design is shown in Figure 2.

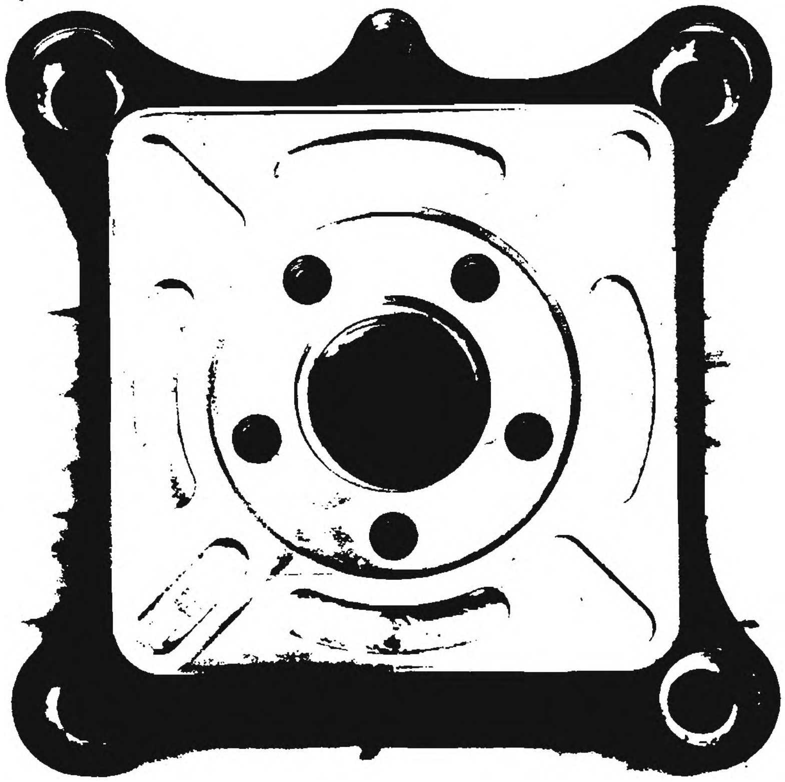

Figure 1. Green piece

Based on the design information the 3-D features listed in Figure 3 were extracted. These were used for matching purposes in the analysis of the 3-D data.

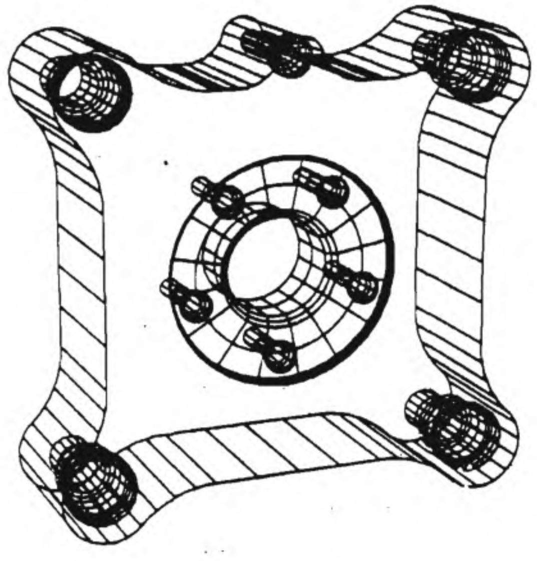

Figure 2. CAGD Design of Green piece

```
% Model definition for Model-1

(setf ModelGP
  (MakeModel
   (Model-Name 'ModelGP)
   (Model-Origin '(0.0 0.0))
   (Model-Feature-Instance-List
       '((Hole865    H1  2.000 2.000 0.0)
         (Hole575    H2  0.375 0.375 0.0)
         (Hole575    H3  0.375 3.625 0.0)
         (Hole575    H4  3.625 0.375 0.0)
         (Hole575    H5  3.625 3.625 0.0)
         (Hole330    H6  2.000 3.835 0.0)
         (Hole250    H7  1.250 1.750 0.0)
         (Hole250    H8  1.563 2.563 0.0)
         (Hole250    H9  2.438 2.563 0.0)
         (Hole250    H10 2.700 1.750  0.0)
         (Hole250    H11 2.000 1.250 0.0)))))
```

Figure 3. 3-D Features Extracted from Model

3. Object Recognition

The system we are developing is based on the notion of *specialization*. This means that we take advantage of any particular information that can be culled from the CAGD shape model. This knowledge is then encapsulated in a special package which provides for the recognition of an object or part of an object. Thus, instead of using a general recognition technique on all parts to be recognized (i.e., a weak method), we produce specially packaged code (i.e., logical sensors) for recognition. These are then instantiated independently as needed, and controlled as logical sensors.

The approach consists of three phases:

1. **Design.** The object is designed using a CAD modeling system (the Alpha_1 CAGD system in our case). This aspect was explained in the previous section.
2. **Derivation of Intrinsic Features.** The recognition strategies are based on matching intrinsic features of the object's shape with those of unknown shapes. A set of intrinsic features are derived from the CAGD system, and includes such features as: genus, surface points, number and placement of holes, color, texture, surface normals, surface curvature, etc. See Henderson and Bhanu [12] for more details on our approach to the use of intrinsic features as the interface between CAD and computer vision systems.
3. **Synthesis of Object Recognition Strategies.** Given a set of intrinsic features for a specific object and knowledge of the representation chosen in the CAGD system (e.g., Constructive Solid Geometry, Generalized Cylinders, or Boundary Representation), plus knowledge of the available recognition techniques and feature detectors, the system will choose and hook together the appropriate recognition code. This is currently done by means of parameters, but eventually will require more expertise in using the knowledge that the system has available.

A straightforward method which we use for generation of recognition strategies is parameterization. The user is required to fill in the blanks for the sensors and algorithms for the particular object, or class of objects, modeled. Obviously, a more automated system is desired for this task and is under investigation [5, 6, 7, 9, 10, 11, 12, 13, 14]. The methodology provides a means for abstracting the specification of a sensor from its implementation along with providing transparency of hardware and software above the implementation level. Alternatively, we are also investigating how to embed knowledge of the algorithms and sensors in the system and to provide a rule base for the decision process. This requires a complex expert system (see [8] for a description of a preliminary system). In either case, the system will eventually be composed of multiple sensors and recognition methods.

4. Split-Level Relaxation Matching

The final part of the system is the matching component. We have recently introduced the notion of "split-level" relaxation [15], and we have applied it here to the problem of labeling 3-D features. One of the first steps in locating an object is to locate its features. We can recognize objects on the basis of *global* features, like number of holes, size of various segments, total area of the segments, perimeter, etc. Alternatively we can also use local features to locate objects; e.g., corners, holes, etc. We look for certain structure with respect to these local features in the image, and if we can find such a structure then we can locate the object.

Both methods have their advantages and disadvantages. A system based on global feature matching is prone to mistakes, particularly when the object is occluded or even partially defective. On the other hand, if the object is completely isolated in the scene under consideration then the method is very straightforward and efficient.

A method based on local features uses only constraints between nearby features. This is useful in many circumstances, when the object is partially occluded and chances are that some of the features would be visible and they can be used to identify the object. However, there are instances when it would be very useful to have global constraints also. And in certain instances it is necessary to use global constraints to identify an object. For example, the object in Figure 4(a), as seen in Figure 4(b) can never be identified using local features alone since all the holes are occluded. However, by using global constraints like the parallelism and perpendicularity of sides, we can easily identify the object.

The approach used here is based on discrete relaxation. However, there are some major philosophical differences. As explained later, we distinguish between different types of constraints during the process of relaxation. Many approaches based on relaxation make use of only the local constraints as opposed to global constraints. They tend to ignore the global constraints because occlusion sometimes prevents their use as well as the high cost of analyzing graphs with high vertex degree. We argue that they can be useful, and under certain circumstances they may be extremely advantageous. The model proposed in this paper uses both types of constraints. Also, with the advent of parallel computers it is imperative to look at the problem again and see if the approaches are suitable for parallel processing or not. This is especially important in light of the fact that the problem at hand has an exponential growth rate. For an example of the use of multiple semantic constraints with stochastic relaxation, see Faugeras and Price [4].

After an image is acquired, the different feature instances are extracted from it. After that all the computation and analysis is bases on these extracted features. Now the main problem is to associate these feature instances with the features of some model. The features thus obtained from the image are related to each other in various ways. For example, the image may have two holes and a corner, and we can have constraints like *above*(Comer,Hole_1), or *bigger*(Hole_1,Hole_2). Obviously we could form many relations, each with different different physical meaning to relate these features. However, we choose only those which are used to define the constraints between the features in the models.

So, the scene can be represented as:

$\mathbf{S} = (\mathbf{F}^s, \mathbf{C}^s)$, where

$\mathbf{F}^s$ is the set of features in the image and

$\mathbf{C}^s = \{ \mathbf{R}_1^s, \mathbf{R}_2^s, \ldots\ldots, \mathbf{R}_k^s \}$, $\mathbf{R}_i^s \subseteq \mathbf{F}^s \times \mathbf{F}^s$.

There is a direct correspondence between the two sets $\mathbf{C}^m$ and $\mathbf{C}^s$. For each relation $\mathbf{R}_i^s$ in $\mathbf{C}^s$ there is an equivalent relation $\mathbf{R}_j^m$ in $\mathbf{C}^m$ which has the same physical interpretation, although the domains of both relations are different.

4.1. Scene Analysis as CLP

Now we formulate scene analysis (SA) as a Consistent Labelling Problem (CLP). The set of *units* which needs to be labelled is the set of feature instances found in the image. For simplicity in analysis here, we assume that the domain of each of these units is the union of the features in all the models. In practice, however it would be the set of features in all models which are of the same type as the unit (feature instance). For example, if the given feature instance is of type *hole*, then only those features in the models of type *hole* can be in its domain. Obviously, the constraints in the models should be included in the constraint set for the CLP. We also have to include the constraints in the image since they also constrain the labelling process.

Features can have the same name in different models. To disambiguate them each feature in the model is prefixed by the model name. For example, feature *Hole* in model *Part* will be referred to as *Part_Hole*. We assume that no two models have the same name and the set of features F_i^m for each model is augmented to remove ambiguities, if any. So, the scene analysis problem in terms of a CLP is:

SA = (U, D, R),

where,

$\mathbf{U = F^s}$, $\mathbf{D = \cup_{i=1}^{n} F_i^m}$, $\mathbf{R = (\cup_{i=1}^{n} C_i^m) \cup C_i}$.

4.2. Network Model for Relaxation

In this section we briefly describe a graph/network model which is the underlying basis for the relaxation process. This is similar to the network model used by Mackworth [17] and others in the sense that the nodes represent the units to be labelled and the constraints are represented by the arcs in the graph. However, there are several differences.

Instead of one graph, we have a set of graphs, one for each of the relations in the image. We also have a set of graphs, corresponding to the object models. This set of graphs is used to represent the constraints in the model. Let G^m, be the composite graph to represent the set of *model graphs* and G^s, denote the same for the *image graph*.

4.2.1. Model Graphs

$G^m = \{ G_1^m, G_2^m, \ldots\ldots, G_n^m \}$, where n is the number of models being considered. Each of the G_i^m also consists of a set of graphs corresponding to the various relations in the constraints.

$G_i^m = \{ G_{i,1}^m, G_{i,2}^m, \ldots\ldots, G_{i,k_i}^m \}$, where k_i is the number of relations in C_i^m.

Each $G_{i,j}^m$ is a graph and can be represented by: $G_{i,j}^m = < F_i^m, E_{i,j}^m >$, where edge (x,y) is in $E_{i,j}^m$, iff $(x,y) \in C_{i,j}^m$. The nodes of the graph are the features in the image which need to be labelled and the arcs represent a relation.

4.2.2. Image Graphs

Unlike the model graphs, we have only one composite image graph, since we are concerned with only one image at a time. However, the composite image graph, G^s, is again composed of a set of graphs, one for each of the relations in the image. So,

$G^s = \{ G_1^s, G_2^s, \ldots\ldots, G_{k_s}^s \}$, where k_s is the number of relations used in the image graphs.

Each of these graphs corresponds to a relation in the image. Therefore,

$G_i^S = < F^S, E_i^S >$, where edge (x,y) is in E_i^S, iff $(x,y) \in R_i^S$.

Although the standard relaxation process works fine for non-occluded scenes, it doesn't work well if the scene is occluded. The basic reason is that if a constraint is missing in the image, it doesn't mean that the constraint actually doesn't hold. It is possible that a constraint is not satisfied because some or all of the associated units are occluded. So the constraints don't have the same discriminating power in occluded scenes. In unoccluded scenes, we use the boundary edges directly to describe the constraints, but this can't be done in occluded scenes.

Also, some of the constraints have to be used or interpreted differently. For example, if a boundary edge X is longer than another edge Y in the model, it doesn't follow that the corresponding edge for X in the image will be longer than the corresponding edge for Y. In fact, if a boundary edge W is longer than another boundary edge Z in an image, no definite conclusion can be made about them from this information alone.

However some of these constraints can be used under certain circumstances. For example, if there are only a few boundary edges which are longer than some absolute amount and we find an edge in the image which is longer than this threshold, then it has to be one of the above edges. But this doesn't constrain the problem enough to reduce the solution set of the other labels.

One way to get around this problem, is to only use local constraints between features like holes, corners, etc. Instead of using constraints between the sides (or boundary edges), we use constraints between the lines between the locations of features. We refer to these vectors as *inter-feature vectors* (or iv's). The advantage of using these iv's instead of the sides is that, unlike the sides, they are either present or absent; i.e., they cannot be partially missing or broken up into different parts because of occlusion. If both the features constituting an iv are present in the image, then the iv is defined, otherwise it is not. We can use the same constraints as before, but now they are between these iv's instead of the sides. Before they were binary constraints, but now they are essentially 4-ary constraints.

However, the constraints still don't have the discriminating power to drive the relaxation process, since the constraint set in the image is incomplete. Initially each unit has all the labels of the model in the label-set. We delete a label at a node if a certain constraint is not satisfied. But now we are not sure if the constraint is unsatisfiable, or is just not satisfied because of occlusion. So, in order to drive the relaxation process we need to seed the label-set of some units and then let relaxation take over. The goal is to get some positive information from the scene and then propagate it. The next two subsections describe how the iv's are created and how the initial seeding is done.

4.3. Creation of Inter-Feature Vectors

An inter-feature vector is defined to be the vector between two feature locations. The magnitude of the vector is the distance between the two feature locations. If there are N features under consideration, then there could be as many as N * (N-1) iv's. This could be too large to be manageable, particularly in an image where there could be a large number of features. To avoid this problem, we connect only the features which are within a certain distance of each other. This distance is not arbitrarily chosen, rather it is computed for each model separately so as to keep the number of iv's small.

4.4. Initial Seeding of Label-Sets

The goal of the seeding process is to reduce the label-sets of some units, drastically if possible so that it will the drive the relaxation engine. This means we need to find the *peculiar* properties of some features, their locations or their relationships. Since we are using the iv's as the basis for the constraints, we look for peculiarities among these iv's. Two such quantities are used here and we found them very useful for the seeding process. They are:

- The **magnitude** of the inter-feature vector (or the distance between the corresponding two feature locations).
- The **angle** between two inter-feature vectors. The two iv's are chosen such that they have exactly one common end point, i.e., they share one feature.

We group these quantities and try to find quantities which are unique. In our case we say a quantity is unique, if it doesn't occur more than a certain number times in the above groups. We determine this by histogramming. Figure 5 gives a brief outline of how this process works. It should be noted that all these things (threshold-distance, inter-feature vectors, histogramming, etc.) are done for the models beforehand. This computation is performed only once.

```
procedure ComputeSpecialDistances;
begin
    Compute the threshold-distance;
    for i:= 1 to No-of-Features do
        for j:=i+1 to No-of-Features do
            If distance(feature[i], feature[j]) <= threshold-distance then
                iv = Make-inter-feature-vector(feature[i], feature[j]);
                Append(IVs, iv)
            endif;
    size := GetSize(IVs, error)
    H := Make-histogram(IVs,size)
    for i:=1 to size do
        if H[i] <= Nunique then
                Append(Special-Length-List, details(i));
        endif;
    Store the Special-Length-List in Model;
end;
```

Figure 5. Computation of Special Distances

Function *details* collects some more information about the inter-feature vectors which correspond to the i^{th} histogram entry (for example, the associated features, etc.). Function *GetSize* computes the size of the histogram. It is computed such that two consecutive entries in the histogram are at least *error* apart. All the special distances are collected in a list and become a part of the model. Special angles are also computed by a similar procedure and are stored in the model. A special-distance is actually not a single quantity, but it has a range associated with it. The idea is that if the distance between two units in the image is in this range, then the two units should correspond to the two features in the model.

We have found that there are a few special lengths and angles for each of the models that are fairly unique in the sense that there is practically no other quantity which is close to these special quantities in terms of their magnitude. This information can be used to seed the label-set of units in the image.

4.5. Labelling Process

The process of labelling starts with *type-checking*. Initially each unit (primitive) in the image is given all the labels of the desired model. Then, if the types of the unit and any of its labels are not consistent then those labels are removed from the label-set. The next step is the seeding process. Before the actual seeding can be done, the inter-feature vectors and the angles are constructed for the image. The distance used is the threshold-distance used by the model. After the lengths and the angles are constructed, they are searched for the special angles and lengths. If any of the distances (or angles) match a special distance (or angle) for the model, then the label-sets of the associated units are updated.

Then control is passed on to the relaxation process, and finally to the backtracking operator. Due the changed nature of the problem, the structure of the relaxation process has been changed considerably, too. In the next section, we give the motivation and the structure of the new version of the relaxation process.

4.6. Split-Level Relaxation Process

The main reason to change the structure of the relaxation process is that there is no way to use positive information in relaxation. For example, if during the seeding process we infer that the primitive X can be either A or B, then we really can't use this information in any way more than if the label-set of X had A and B without seeding. Relaxation treats all the nodes equally and is not able to use the information from seeding, which is very powerful and positive information. As a matter of fact, it is possible for node X to lose all its labels, during relaxation just because it lacks support at some other primitive which may not even belong to the model.

We divide the nodes (primitives) into two groups. The nodes whose labels are fixed during the seeding process, are called the *strong* nodes and others are called *weak* nodes. The *strong* nodes signify positive information. The strong nodes, always remain strong, while the weak nodes may be elevated to strong status. During relaxation, a strong node can affect the label-set of another (either strong or weak) node. This prevents the nodes which are not really a part of the model from affecting the label-sets of other nodes. Those units which survive the first iteration, i.e., whose label-set is not empty at the end of the iteration, are then changed to *strong* nodes. Those nodes which do not survive the first iteration, are not considered further and are not considered to be not part of the model. After that, the relaxation process works as usual.

4.6.1. A Second Approach

Another way to solve the problem of *weak* nodes deleting the labels of *strong* nodes is to allow a label at a node to remain, if there is at least one support for it from any one other node. (In standard discrete relaxation, a label remains iff it has support from all neighboring nodes.) This is motivated by the fact that some of the constraints may be unsatisfied due to occlusion. It should be noted, however, that there has to be support for the label in each of the possible graph types. This is fairly effective, but takes more time, since the labels are deleted after all the constraints are processed. The effect of change can only be perceived during the next iteration. In split-level relaxation, we delete a label once it is determined that there is no support, and the change can be used by the next constraint during the same iteration.

4.7. Implementation

Most of the above ideas were implemented in PSL (Portable Standard Lisp) [18]. The compiled code was run on a VAX-8600. The actual runtimes can be vastly improved if implemented on a lisp machine like the Symbolics 3600.

The system works in two phases. In the first phase, all the models are input and the threshold distance, special lengths and angles are determined and stored in the model structure. This is only a one-time cost and is similar to the training phase of Local Feature Focus [3]. The input is at a somewhat intermediate level of description. What is input to the program is a list of local features, their locations and the constraints between them. The different types of constraints used are *parallel*, *perpendicular*, *adjacent*, *longer-than*, etc.

In the second phase, an image description is given along with the set of constraints associated with the local features in it. We also give a model to be searched for in the image. We only check for node and arc consistency. The AC-3 algorithm as defined by Mackworth [17] is used for enforcing the consistency. Although, we look for a single object in an image, it would be be a straight-forward extension to look for multiple objects in the same image. The image description is similar to that for the models.

At the end of relaxation process, control is passed on to a backtracking procedure which does further consistency checks and lists out all the solutions. The output is a listing of local features in the image and the corresponding features in the object. If the label-set of a primitive is empty, it is labelled as *unknown*.

5. An Example

Figure 6 shows a range data view of an actual milled version of the piece modeled in Section 2. The features extracted from that data are given in Figure 7. These were fed to the split-level relaxation matcher and the results are shown in Figure 8.

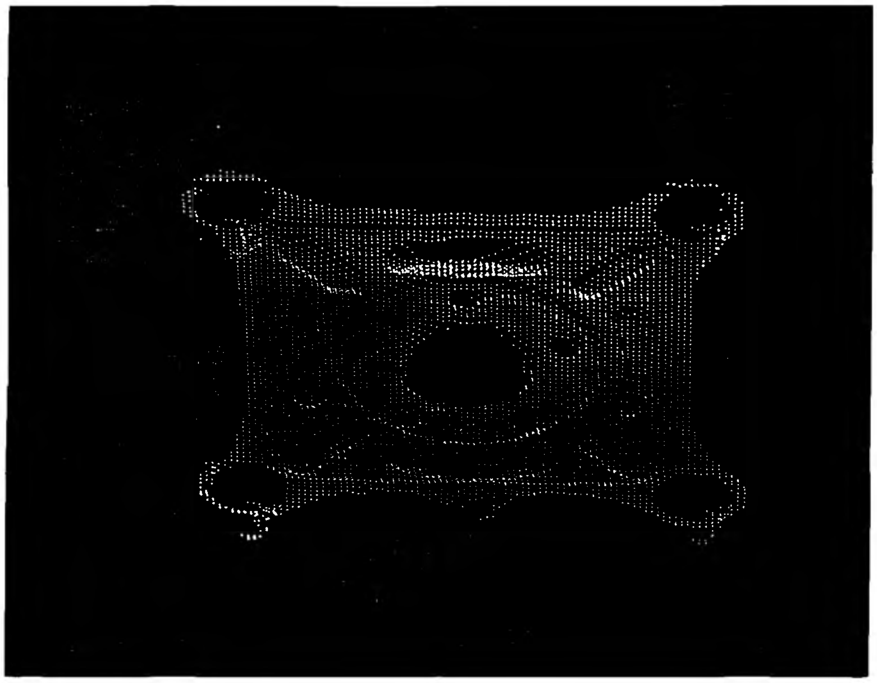

Figure 6. Range Data View of Green Piece

```
% Definition for Image-1

(setf Image
  (MakeImage
   (Image-Name 'ImageGP)
   (Image-Origin '(0.0 0.0))
   (Image-Feature-Instance-List
       '((Hole575    Hole1    1.520 -3.652 -0.487)
         (Hole575    Hole2    1.558 -0.394 -0.465)
         (Hole250    Hole3    0.800 -2.255 -0.943)
         (Hole250    Hole4    0.528 -1.363 -1.376)
         (Hole865    Hole5    0.149 -1.958 -1.294)
         (Hole250    Hole6    0.151 -2.762 -1.299)
         (Hole250    Hole7   -0.233 -1.367 -1.503)
         (Hole250    Hole8   -0.484 -2.204 -1.634)
         (Hole575    Hole9   -1.330 -0.344 -2.038)
         (Hole575    Hole10  -1.375 -3.603 -2.039)))))
```

Figure 7. Features Extracted from Range Data

```
%% Output for Split-level Relaxation

Input File is params.sl

Model definition  file is modelgp.sl
Model constraints file is modelgpC.sl
Image definition  file is imgp.sl
Image constraints file is imgpC.sl
Optimal distance = 1.10105

Initial statistics
   Total Number of Nodes = 10
   Number of nodes with one label = 1
   Average number of labels/node  = 3.0

Iteration Number = 1 Time = 51 ms
   Number of nodes with one label = 3
   Average number of labels/node  = 2.5

Iteration Number = 2 Time = 1122 ms
   Number of nodes with one label = 5
   Average number of labels/node  = 1.5

Iteration Number = 3 Time = 238 ms
   Number of nodes with one label = 6
   Average number of labels/node  = 1.4

Iteration Number = 4 Time = 238 ms
   Number of nodes with one label = 6
   Average number of labels/node  = 1.4

Total time for ARC Consistency = 1683 ms

There is only one labelling for the above

   Primtive = HOLE10   : Label = H2
   Primtive = HOLE9    : Label = H3
   Primtive = HOLE8    : Label = H7
   Primtive = HOLE7    : Label = H8
   Primtive = HOLE6    : Label = H11
   Primtive = HOLE5    : Label = H1
   Primtive = HOLE4    : Label = H9
   Primtive = HOLE3    : Label = H10
   Primtive = HOLE2    : Label = H5
   Primtive = HOLE1    : Label = H4
```

Figure 8. Matching Results

6. Conclusions

This approach to model-based 3-D data analysis shows great promise. Moreover, the inherent capability to automate the generation of recognition and manipulation code will make systems such as the one discussed here a very important tool in the automation industry.

Clearly the success of the method depends on the special-lengths and angles. If they are occluded, then the method may take a longer time than usual. During the process, it might lose all the labels in which case it would not recognize the object. Or it may reduce the label-sets of some nodes and leave the rest of the work to the backtracking procedure, which can be expensive.

However, if there are enough *special* distances and angles, then even if some are missing, the method will work very well. Our experience is that this will work fine, if the label-sets of two or three nodes can be reduced to about 2 or 3 labels. Also, it is not unusual to find about 4 or 5 special distances and angles. If they are not enough, we could increase the threshold distance (for connection) to include more inter-feature vectors, thereby increasing the possibility of finding more special distances and angles. Of course, this increases the cost of computation, both for the model and the image. The positive side of this is the fact that, these computations can be done ahead of time, more than once if necessary to arrive at an optimal set of special distances and angles. Once this is done, it is a part of the model of the object and need not be recomputed.

Also, this takes care of a certain amount of error in both the model and the image. As pointed out before, the special distances have a range associated with them. So, if there is any error the initial seeding is not affected. This makes the system more robust.

References

[1] Ballard, D.H. and C.M. Brown.
Computer Vision.
Prentice Hall, New York, 1982.

[2] Barrow, Harry and Jay Tennenbaum.
Recovering Intrinsic Scene Characteristics from Images.
Technical Report 157, SRI International, April, 1978.

[3] Robert C. Bolles and Ronald A. Cain.
Recognizing and Locating Partially Visible Objects : The Local-Feature-Focus Method.
The International Journal of Robotics Research 1(3):57-82, Fall, 1982.

[4] Faugeras, Olivier and Keith Price.
Semantic Description of Aerial Images Using Stochastic labeling.
IEEE Transactions on Pattern Analysis and Machine Intelligence :633-642, November, 1981.

[5] Hansen, C., T.C. Henderson, Esther Shilcrat and Wu So Fai.
Logical Sensor Specification.
In *Proceedings of SPIE Conference on Intelligent Robots*, pages 578-583. SPIE, November, 1983.

[6] Henderson, T.C., E. Shilcrat and C. Hansen.
A Fault Tolerant Sensor Scheme.
Computer Science UUCS 83-003, University of Utah, November, 1983.

[7] Henderson, T.C., E. Shilcrat and C.D. Hansen.
A Fault Tolerant Sensor Scheme.
In *Proceedings of the International Conference on Pattern Recognition*, pages 663-665. August, 1984.

[8] Henderson, T.C., Bir Bhanu, C.D. Hansen and E. Shilcrat.
ASP: A Sensor Performance and Evaluation System.
In *Proceedings of Pecora IX*, pages 201-207. October, 1984.

[9] Henderson, Thomas C., Bir Bhanu and Chuck Hansen.
Distributed Control in the Multisensor Kernel System.
In *Proceedings SPIE Conference on Intelligent Robots*, pages 253-255. Cambridge, Massachusettes, November, 1984.

[10] Henderson, T.C. and C.D. Hansen.
A Kernel for Multi-sensor Robotic Systems.
In *Proceedings of the CAD/CAM, Robotics and Automation Institute and Conference*, pages to appear. February, 1985.

[11] Henderson, T.C., C.D. Hansen, and Bir Bhanu.
The Specification of Distributed Sensing and Control.
Journal of Robotic Systems :to appear, 1985.

[12] Henderson, T.C., Bir Bhanu and Chuck Hansen.
"Intrinsic Characteristics as the Interface between CAD and Machine Vision Systems.
Pattern Recognition Letters :to appear, 1985.

[13] Henderson, T.C., Chuck Hansen and Wu So Fai.
Organizing Spatial Data for Robotic Systems.
Computers in Industry :to appear, 1985.

[14] Henderson, T.C., Chuck Hansen and Bir Bhanu .
A Framework for Distributed Sensing and Control.
In *Proceedings of IJCAI 1985*, pages 1106-1109. Los Angeles, CA, August, 1985.

[15] Henderson, Thomas and Ashok Samal.
Split-Level Relaxation.
Technical Report UUCS-85-113, University of Utah, Department of Computer Science, November, 1985.

[16] Horn, B.K.P.
Obtaining Shape from Shading Information.
In P. Winston (editor), *The Psychology of Computer Vision*, pages 115-155. McGraw-Hill, New York, 1970.

[17] Alan K. Mackworth.
Consistency in Network of Relations.
Artificial Intelligence 8:99-118, 1977.

[18] The Utah Symbolic Computation Group.
The Portable Standard LISP users Manual
Department of Computer Science, 1983.

[19] Rosenfeld, A. and A. Kak.
Digital Picture Processing.
Academic Press, New York, NY, 1976.

[20] Witkin, A. P.
Recovering Surface Shape and Orientation from Texture.
Artificial Intelligence 17:17-45, 1981.