Robert C. Bolles
Ronald A. Cain
Robotics Department
Computer Science and Technology Division
SRI International
Meno Park
California 94025

Recognizing and Locating Partially Visible Objects: The Local-Feature-Focus Method

Abstract

A new method of locating partially visible two-dimensional objects is presented. The method is used to locate complex industrial parts that may contain several occurrences of local features, such as holes and corners. The matching process utilizes clusters of mutually consistent features to hypothesize objects and also uses templates of objects to verify these hypotheses. The technique is fast because it concentrates on key features that are automatically selected on the basis of a detailed analysis of computer-aided design (CAD) models of the objects. The automatic analysis applies general-purpose routines for building and analyzing representations of clusters of local features that could be used in procedures to select features for other locational strategies. These routines include algorithms for computing the rotational and mirror symmetries of objects in terms of their local features.

Key terms: object recognition, two-dimensional part location, industrial vision, symmetry, computer vision, planning.

1. Introduction

One of the factors inhibiting the widespread development of industrial automation is the robot's inability to acquire a part from storage and present it to a workstation in a known position and orientation. Industrial vision systems that are currently available, like Machine Intelligence Corporation's VS-100 vision system (Kinnucan 1981), Bausch and Lomb's OMNICON (Bausch and Lomb 1976), and Automatix's vision module (Reinhold and VanderBrug 1980) can recognize and locate isolated parts against a contrasting background only. These systems recognize binary patterns by measuring global features of regions, such as area, elongation, and perimeter length, and then comparing these values with stored models. Many tasks fit the constraints of these systems quite naturally or can easily be engineered to do so. However, there are also many important tasks in which it is difficult or expensive to arrange for the parts to be isolated and completely visible. In this paper, we describe a technique for identifying and locating partially visible objects on the basis of two-dimensional models.

Tasks that involve the location of partially visible objects range from the relatively easy, such as locating a single two-dimensional object, to the extremely difficult, such as locating three-dimensional objects jumbled together in a bin. In this paper, we concentrate on tasks that are two-dimensional in the sense that the uncertainties in the location of an object are in a plane parallel to the image plane of the camera. This restriction implies a simple one-to-one correspondence between (1) sizes and orientations in the image and (2) sizes and orientations in the plane of the objects.

This class of two-dimensional tasks can be partitioned into four subclasses that are defined, with respect to the complexity of the scene, as follows:

A portion of one object
Two or more objects that may touch one another
Two or more objects that may overlap one another
One or more objects that may be defective

The work reported in this paper was supported by the National Science Foundation under Grant No. DAR-8023130.

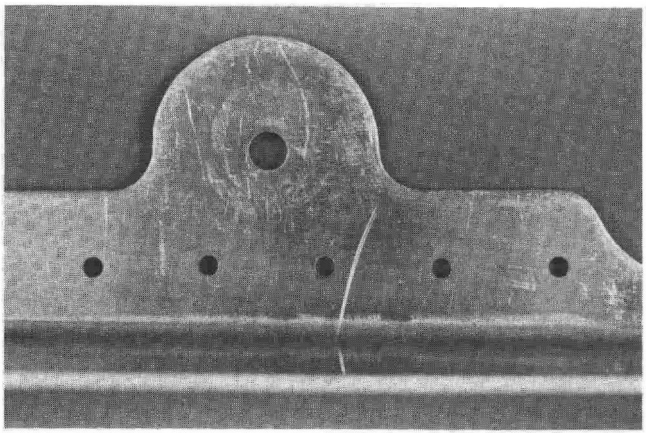
The International Journal of Robotics Research
Vol. 1, No. 3, Fall 1982
0278-3649/82/030057-26 $05.00/0

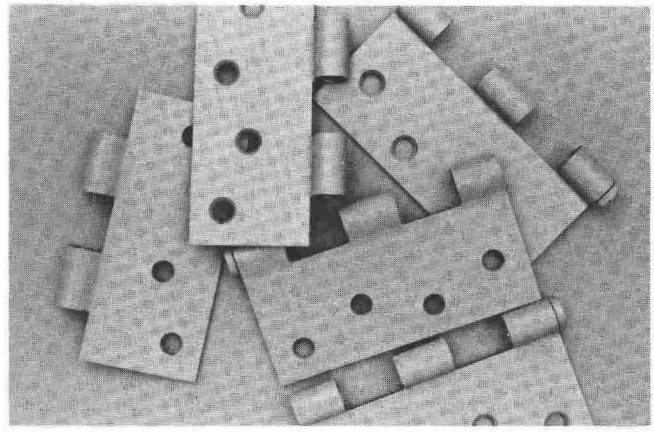
Fig. 1. Partial view of an aircraft frame member.

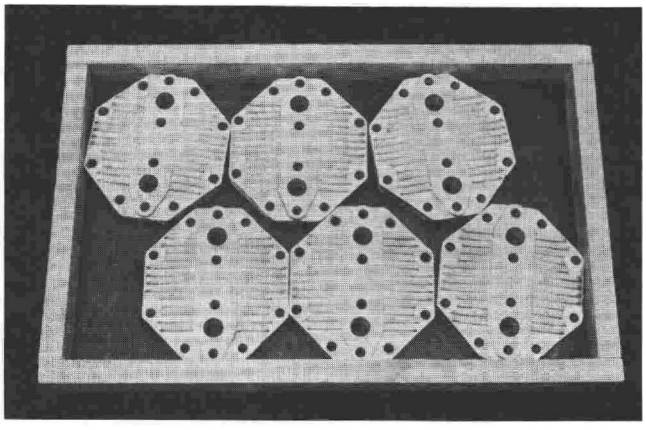
Fig. 2. Mutually touching parts in a tray.

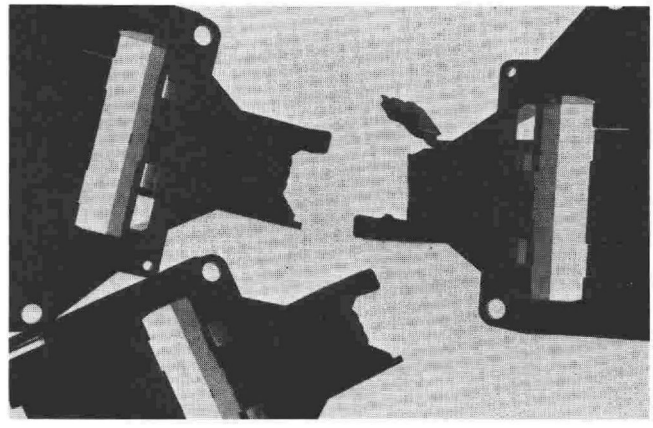
Fig. 3. Overlapping parts.

Fig. 4. Three parts, one of which is defective.









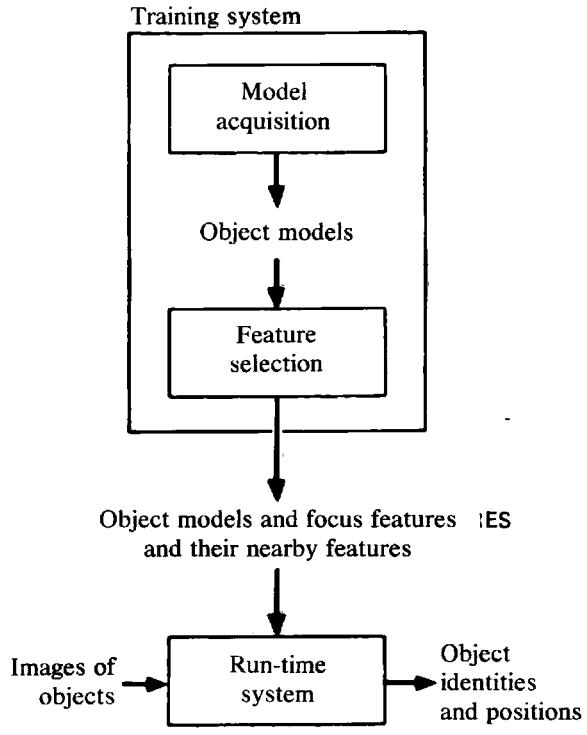
This list is ordered roughly by the amount of effort required to recognize and locate the objects. Examples of these subclasses are shown in Figs. 1–4.

Figure 1 shows a portion of an aircraft frame member. A typical task might be to locate the pattern of holes for mounting purposes. Since only one frame member is visible at a time, each feature appears at most once, which simplifies feature identification. If several objects can be in view simultaneously and can touch one another (as in Fig. 2), the features may appear several times. Boundary features such as corners may not be recognizable, even though they are in the picture, because the objects are in mutual contact. If the objects can lie on top of one another (as in Fig. 3), even some of the internal holes may be unrecognizable if they are partially or completely occluded. Finally, if one of the objects is defective (as in Fig. 4), its features are even less predictable and hence harder to find.

Since global features are not computable from a partial view of an object, recognition systems for these more complex tasks must be programmed to recognize either local features, such as small holes and corners, or extended features, such as a large segment of an object's boundary. Both types of features, when found, place constraints on the positions and orientations of their objects. Extended features are in general computationally more expensive to find, but they provide more information because they tend to be less ambiguous and more precisely located.

Given a description of an object's features, the

Fig. 5. Top-level block diagram of the LFF method.



time the system requires for matching this description with a set of observed features appears to increase exponentially with the number of features (Karp 1972). Therefore, practical applications have been restricted to relatively simple tasks like identifying the holes in Fig. 1. The multiplicity of features in Figs. 2–4 precludes the straightforward application of any simple matching technique.

Three approaches have been explored for tasks involving a large number of features. The first is to locate a few extended features instead of many local ones (see for example, papers by Perkins [1978] and Ballard [1981]). Even though it costs more to locate extended features, the reduction in the combinatorial "explosion" is often worth it. The second approach is to start by locating just one feature and then use it to restrict the search areas for nearby features (Tsuji and Nakamura 1975; Holland 1976). Concentrating on one feature may be risky, but here, too, the reduction in the total number of features to be considered is often worth it. The third approach is to sidestep the problem by hypothesizing massively parallel computers that can perform matching in linear time. Examples of these approaches include graph-matching (Ambler et al. 1973; Bolles 1979*a*), relaxation (Zucker and Hummel 1979; Barnard and Thompson 1980), and histogram analysis (Duda and Hart 1972; Tsuji and Matsumoto 1978; Ballard 1981). The advantage of these approaches is that their decisions are based on all the available information.

Although parallel computers surely will be available sometime in the future, in this paper we restrict our attention to recognition methods for sequential computers. The basic principle of the local-feature-focus (LFF) method is to find one feature in an image, referred to as the *focus feature*, and use it to predict a few nearby features to look for. After finding some nearby features, the program uses a graph-matching technique to identify the largest cluster of image features matching a cluster of object features. Since the list of possible object features has been reduced to those near the focus feature, the graph is relatively small and can be analyzed efficiently.

The key to the LFF method is an automatic feature-selection procedure that chooses the best focus features and the most useful sets of nearby features. This automatic-programming capability makes possible quick and inexpensive application of the LFF method to new objects. As Fig. 5 shows, the *training* process, which includes the selection of features, is performed once, and the results are used repeatedly.

In this paper, we describe the run-time system first so as to clarify the requirements of the training system. After describing the training system, we conclude with an evaluation of the LFF method and a discussion of possible extensions.

2. Run-Time System

The run-time phase of the LFF system acquires images of partially visible objects and determines their identities, positions, and orientations. This processing occurs in four steps:

1. Reading task information
2. Locating local features

Fig. 6. Half of a metal door hinge.

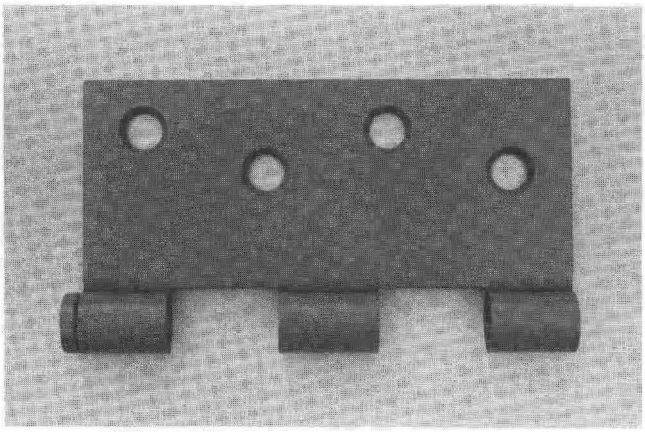


Fig. 7. Binary image of the hinge part.

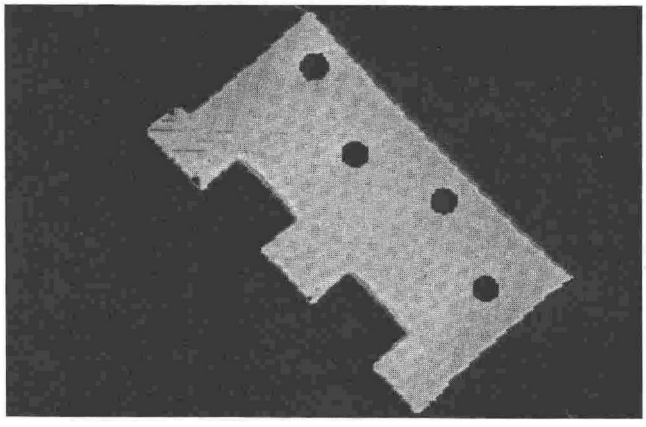
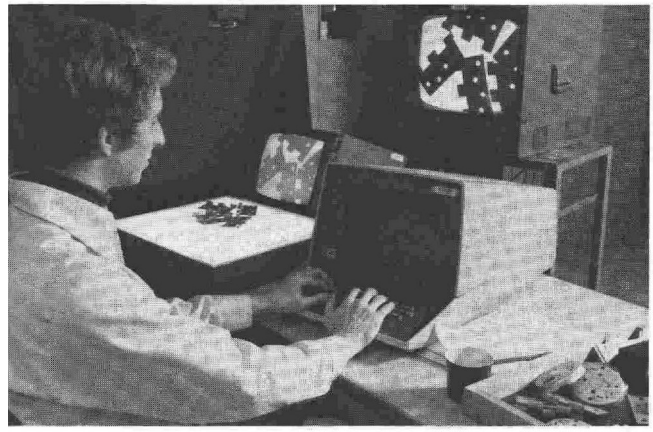


Fig. 8. The LFF workstation.



3. Hypothesizing objects
4. Verifying hypotheses

The first step, as indicated in Fig. 5, is to enter the object models, together with the list of focus features and their nearby features. Then, for each image, the system locates all the potentially useful local features, forms clusters of them to hypothesize object occurrences, and finally performs template matches to verify these hypotheses. A simple task will be used to illustrate the basic processing of these steps. (Later, we will consider a few more difficult tasks that demonstrate additional capabilities of the system.) The initial assignment is to locate the object in Fig. 6, which is half of a metal door hinge, in the image in Fig. 7.

The examples in this paper were produced by an implementation of the LFF system in which the run-time system is executed on a PDP-11/34 minicomputer and the feature selection runs on a VAX-11/780. The run-time software is a modified version of the SRI vision module software (Gleason and Agin 1979), which recognizes and locates isolated, completely visible objects in binary images. Figure 8 shows the LFF workstation. The camera is a General Electric TN2500 camera, which provides a 240-by-240-pixel array. It views objects placed directly beneath it on a light table. Gray-scale and binary images are displayed on monitors, and the user interacts with the system through a Tektronix display. The fact that the current implementation locates local features in binary images is due to the ready availability of the appropriate software. However, because the techniques for selecting and matching clusters of local features are independent of the manner in which the local features are detected, they can be applied directly to the task of locating objects in gray-scale images or in range data, given local feature detectors for such data.

2.1. Task Information

The task information consists of the following:

Statistical descriptions of local features
Analytic descriptions of objects
A strategy for locating the objects

The local features are defined in terms of their ap-

pearances in images. The objects and the locational strategy are defined with respect to these local features.

In the current implementation there are two types of features, regions and corners. A *region* is described with respect to its color (black or white), area, and axis ratio (the ratio of its minor and major axes). A *corner* is characterized by the size of its included angle. Each property of a feature is assigned an expected value and, if appropriate, a variance about that value. Variances are important because the system can use them to locate the features efficiently. In particular, the system can use a feature-vector pattern-recognition routine, such as the nearest-neighbor algorithm used by the SRI vision module, to identify features.

Although the object descriptions are used mainly at run time to verify hypotheses, we will discuss them before describing the strategies because they contain a list of local features for each object. The features in these lists are referred to as *object* or *model* features, as distinguished from *image* features, (i.e., those in images). Each object feature is assigned a unique name and is described in terms of its type and its position and orientation with respect to the object. The positions and orientations have associated variances that are designed to model a combination of the imprecision associated with manufacturing the object and the imprecision associated with locating features in an image. The system uses these variances to decide, among other things, whether the distance between two image features is sufficiently close to the distance between two corresponding object features to be called a match.

In the current implementation, a description of an object also contains a description of its boundary. The boundary is represented by a list of points defining a sequence of line segments that approximate the boundary of the object. Once an object has been hypothesized at a particular position and orientation in an image, the boundary is rotated and translated to that location and checked against the image.

The strategy portion of the task information, which is a list of focus features and their nearby features, is a reformulation of the object descriptions into a procedure-oriented list of features for finding the objects. For each focus feature there is a list of nearby features that, if found, can be used to identify the focus feature and establish the position and orientation of the object. The description of a nearby feature includes its type, its range of distances from the focus feature, its range of orientations with respect to the focus feature (if it has an inherent orientation), and a list of the object features, any one of which it might be. Thus, after finding an occurrence of a focus feature, the program simply runs down the appropriate list of nearby features and looks in the image for features that satisfy the specified criteria. As it finds matching features it builds a list of possible object-feature-to-image-feature assignments. This list is transformed into a graph that is analyzed by an algorithm for finding the largest completely connected subgraph. The subgraph corresponds to a set of assignments that can be used to hypothesize an object.

Figure 9 contains the information produced by the feature-selection phase for the task of locating the hinge part shown in Fig. 6. Three types of local features are described: holes, A-type corners, and B-type corners. An A-type corner is a concave corner; the hinge occupies three-quarters of a small circle centered on the vertex. A B-type corner is a convex corner. The hinge, according to its object description, has four holes, four A-type corners, and eight B-type corners. The object-specific names for the four holes are hole 1, hole 2, hole 3, and hole 4. (See Fig. 14 for a definition of the object-feature names used in Fig. 9.)

The best focus feature for locating a hinge, as noted in the strategy, is a hole. The second best focus feature is an A-type corner; the third best is a B-type corner. According to the locational strategy, once it has found a hole the system should look for seven nearby features, the first of which is a B-type corner that is between 0.527 and 0.767 in. from the hole and whose orientation with respect to a vector from the hole to the corner is between 155.62 and 180.00° or between −180.00 and −160.03°. If any B-type corners are found that meet these criteria, they are likely to be object feature B6. In this example (Fig. 9), by checking all pairs of holes and B-type corners, the training system has narrowed down from eight to one the list of possible object features for this type of corner with respect to a hole. This

Fig. 9. Task information for the hinge part.

Local-feature descriptions

HOLE	type:	REGION	
	color:	WHITE	
	area:	0.071	<variance .001>
	axisRatio:	1.000	<variance .082>
A	type:	CORNER	
	chord length:	12	
	include angle:	270.0	<variance 40.703>
B	type:	CORNER	
	chord length:	12	
	include angle:	90.0	<variance 40.703>

Object description

HINGE	(boundary list:	(0.000,0.000,	0.000,4.000,	2.230,4.000,
		2.230, 3.310,	1.700,3.310,	1.700,2.430,
		2.230,2.430,	2.230,1.625,	1.700,1.625,
		1.700,0.745,	2.230,0.745,	2.230,−0.13,
		1.700,−0.13,	1.700,0.000,	0.000,0.000))

object feature	type	(x,y)	Orientation	xy variance	Orientation variance
Hole 1	HOLE	0.440,0.475	—	0.004	0
Hole 2	HOLE	0.790,1.475	—	0.004	0
Hole 3	HOLE	0.440,2.475	—	0.004	0
Hole 4	HOLE	0.790,3.475	—	0.004	0
a1	A	1.700,1.625	135.0	0.010	21.447
a2	A	1.700,0.745	−135.0	0.010	21.447
a3	A	1.700,2.430	−135.0	0.010	21.447
a4	A	1.700,3.310	135.0	0.010	21.447
b1	B	0.000,4.000	−45.0	0.010	21.447
b2	B	2.230,4.000	−135.0	0.010	21.447
b3	B	2.230,3.310	135.0	0.010	21.447
b4	B	2.230,2.430	−135.0	0.010	21.447
b5	B	2.230,−0.13	135.0	0.010	21.447
b6	B	0.000,0.000	45.0	0.010	21.447
b7	B	2.230,0.745	−135.0	0.010	21.447
b8	B	2.230,1.625	135.0	0.010	21.447

reduction demonstrates the benefits possible from training-time analysis. In the next few sections we shall use the task information in Fig. 9 to illustrate run-time processing.

2.2. Local-Feature Location

The goal of the feature-location step, which is the first type of processing applied to a new image, is to find features in the image that match the local-feature descriptions in the task information. The current implementation locates all the features it can and passes a list of them to the hypothesis-generation step. The assumption is that in the future there will be special-purpose hardware processors that can locate efficiently all local features in an image. If features are relatively expensive to locate, however, feature detection can be integrated into the hypothesis-generation step to minimize processing time.

Fig. 9 (cont'd)

Focus features and their nearby features

HOLE
possible object features: (hole 1 hole 2 hole 3 hole 4)

nearby feature	distance range	orientation range	possible object features:
B	0.527,0.767	155.62,−160.03	**b6**
A	0.801,1.041	107.32, 145.25	**a1**
B	0.828,1.068	149.83,−172.63	b1
HOLE	1.019,1.099	— , —	hole 1 hole 2 hole 3 hole 4
A	1.140,1.380	−149.83,−116.02	a2 a3
B	1.467,1.707	−166.85,−135.28	b1 b2
A	1.050,1.290	−113.39, −78.67	a2

A
possible object features: (a1 a2 a3 a4)

nearby feature	distance range	orientation range	possible object features:
B	0.325,0.725	100.21, 167.93	b3 b8
B	0.325,0.725	−168.62,−100.96	b4 b7
B	0.820,1.220	171.27,−143.20	b2 b5 b8
A	0.610,1.010	109.21, 160.72	a3
HOLE	1.391,1.631	— , —	hole 3
A	0.610,1.010	−160.37,−108.86	a1
HOLE	1.050,1.290	— , —	hole 2 hole 3
HOLE	1.173,1.413	— , —	hole 1 hole 2 hole 3 hole 4

B
possible object features: (b1 b2 b3 b4 b5 b6 b7 b8)

nearby feature	distance range	orientation range	possible object features:
B	0.490,0.890	−163.29,−106.69	b3 b5 b8
A	0.670,1.070	−122.21, −72.71	a1 a4
HOLE	0.828,1.068	— , —	hole 4
B	0.680,1.080	−69.56, −20.40	b4 b7
B	0.490,0.890	109.84, 159.34	b2 b4 b7
A	0.820,1.220	81.59, 127.14	a2 a3
HOLE	0.527,0.767	— , —	hole 1
HOLE	1.467,1.707	— , —	hole 2 hole 3 hole 4
A	0.325,0.725	12.03, 79.76	a2 a3
B	1.490,1.890	−63.60, −27.10	b4 b7

The current system locates regional features, such as the holes described in Fig. 9, by finding regions in the binary image whose properties are sufficiently close to the nominal values. The system locates corners by moving a jointed pair of chords around the boundaries and comparing the angle between the chords to the angles defining the different types of corners. Figure 10 shows the corners located in the image in Fig. 7. This method of finding corners is only one of many possible methods. It was chosen for its simplicity and speed. It encounters difficulties with rounded corners and its precision is influenced by image quantization, but we have found it to be an effective way of finding corners.

The product of the feature-location step is a list of local features found in the image. Given the image-feature numeration in Fig. 11, the list of features in Fig. 10 will be the one shown in Fig. 12. At this

Fig. 10. Corners detected in Fig. 7.

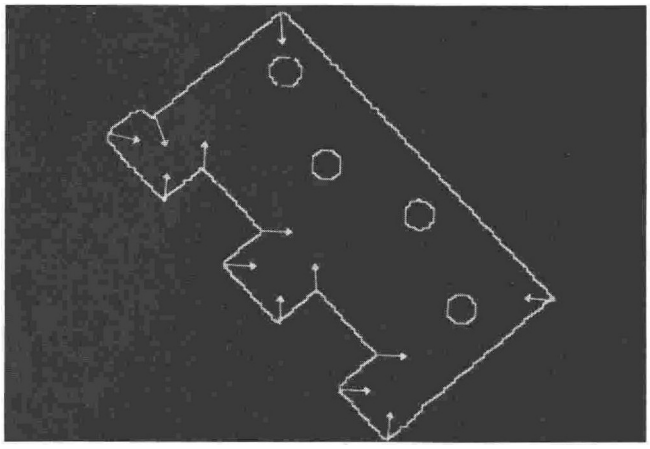


Fig. 11. Image-feature numbers.

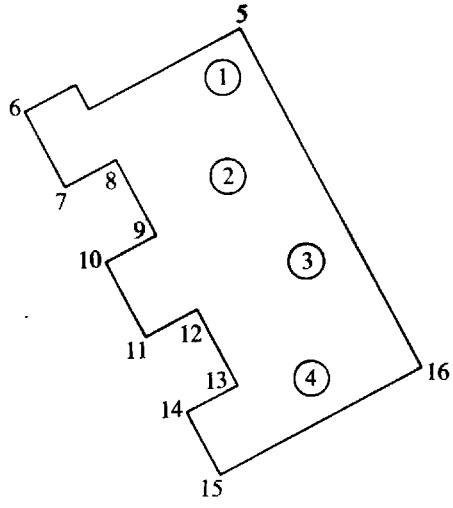
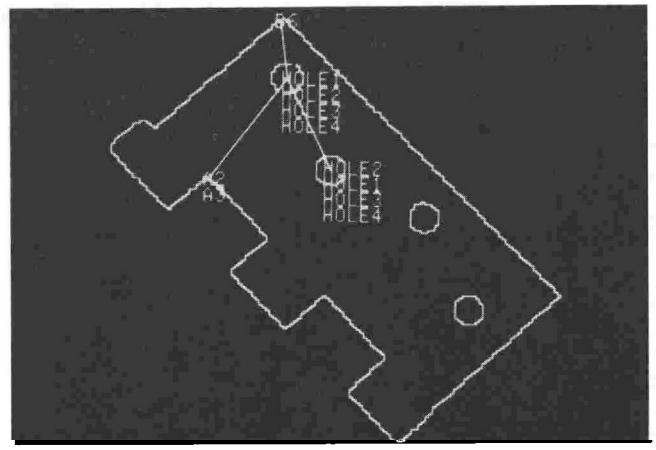


Fig. 12. List of local features found in Fig. 7 and their types.

1. hole	2. hole	3. hole	4. hole
5. B	6. B	7. B	8. A
9. A	10. B	11. B	12. A
13. A	14. B	15. B	16. B

Fig. 13. Nearby features found around a hole and their lists of the possible object features.



stage in the processing, the program has not yet determined the object-feature names—it has just determined their types (i.e., hole, A-type corner, or B-type corner). To assign object-feature names, the program has to analyze the relative positions and orientations of the features. This processing is described in Section 2.3.

2.3. Hypothesis Generation

The goal of the hypothesis-generation step is to generate good hypotheses as fast as possible. As usual, there is a trade-off between good and fast. The optimum procedure is a function of the cost of verifying a hypothesis relative to the cost of generating one. The LFF system has been developed on the premise that the cost of adequate verification is too large to be ignored. Therefore, it is important to generate the best hypotheses possible.

The run-time system hypothesizes objects by recognizing clusters of image features that match clusters of object features. To find these clusters and avoid as much of the combinatorial explosion as possible, the system locates one feature around which it tries to "grow" a cluster. If this does not lead to a hypothesis, the system seeks another focus feature for a renewed attempt.

In the example of the hinge, the feature-selection step rates holes as the best focus features. Consequently, the system looks first for holes. For the image in Fig. 7, the system starts with the uppermost hole. Having selected one, the system then searches the list of local features for those that fit the specifications for features near holes. Figure 13 shows the nearby features found around the hole.

Fig. 14. Local features of a hinge part.

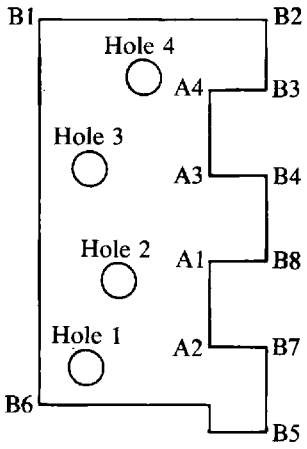
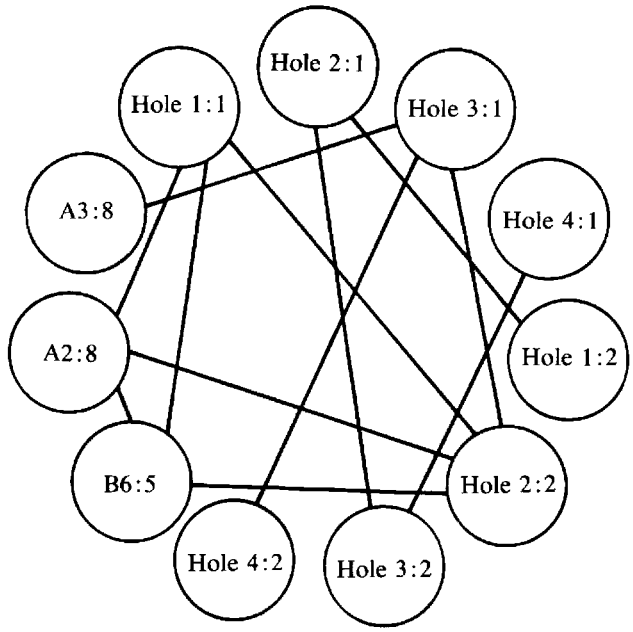


Fig. 15. List of object-feature-to-image-feature assignments.

Hole1 to image-feature 1	Hole2 to image-feature 1
Hole3 to image-feature 1	Hole4 to image-feature 1
Hole1 to image-feature 2	Hole2 to image-feature 2
Hole3 to image-feature 2	Hole4 to image-feature 2
B6 to image-feature 5	A2 to image-feature 8
A3 to image-feature 8	

Fig. 16. Graph of pairwise-consistent assignments.



Beside each feature is a list of the object features it could be (see Fig. 14 for definitions of the object features). Figure 15 lists this information in terms of possible object-feature-to-image-feature assignments.

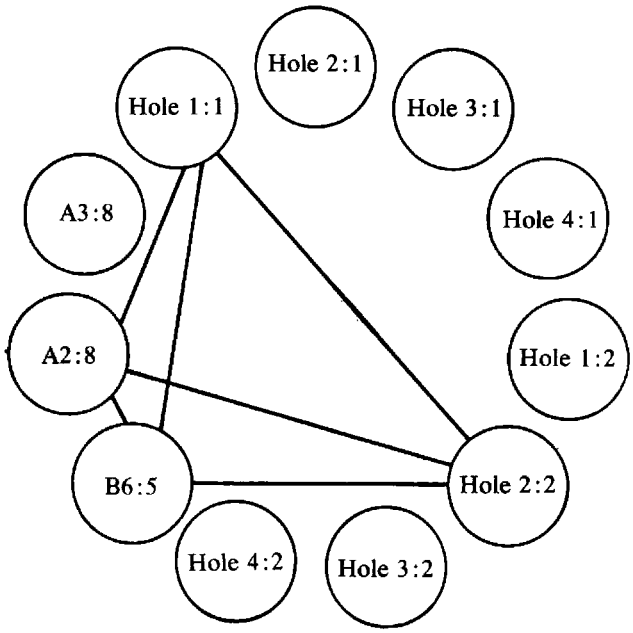
Given the list of possible assignments in Fig. 15, the run-time system uses a graph-matching technique to locate the largest clusters of mutually consistent assignments. In the current implementation, the technique being used is a maximal-clique algorithm (see the Appendix). To apply this technique, the system transforms the list in Fig. 15 into the graph structure in Fig. 16. Each node in the graph represents a possible assignment of an object feature to an image feature. Two nodes in the graph are connected by an arc if the two assignments they represent are mutually consistent. To be mutually consistent, a pair of assignments must meet the following criteria:

- The two object features must not refer to the same image feature.
- The two image features must not refer to the same object feature.
- The two image features must refer to object features that are part of the same object.
- The distance between the two image features must be approximately the same as the distance between the two object features.
- The relative orientations of the two image features with respect to the line joining their centers must be approximately the same as the relative orientations of the two object features.

The assignment hole 1 to image-feature 1 is mutually compatible with hole 2 to image-feature 2, because image-feature 1 is approximately the same distance from image-feature 2 as object-feature hole 1 is from object-feature hole 2 in the analytic model. Hole 1 to image-feature 1 is not mutually compatible with Hole 2 to image-feature 1, because both object features are assigned to the same image feature. Similarly, hole 1 to image-feature 1 is not mutually compatible with Hole 1 to image-feature 2, because one object feature cannot be assigned to two different image features.

Once the graph has been constructed, the graph-matching technique is used to extract the largest sets

Fig. 17. Largest maximal clique for the graph in Fig. 16.



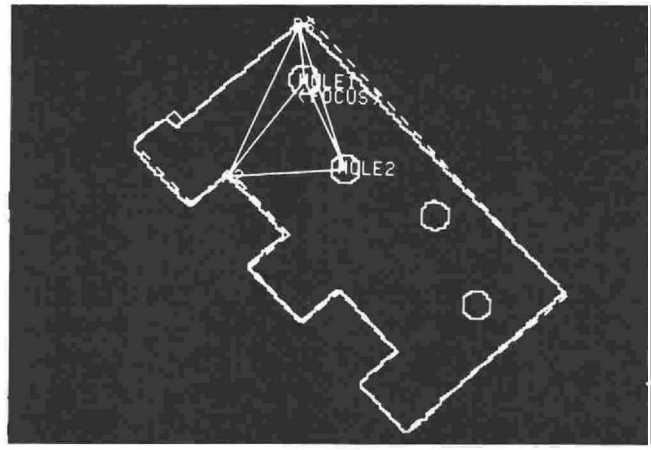
of mutually consistent clusters of nodes. The largest completely connected subgraph (i.e., the largest maximal clique) for the hinge is shown in Fig. 17. It contains four nodes representing the following assignments:

Hole 1 to image-feature 1
Hole 2 to image-feature 2
A2 to image-feature 8
B6 to image-feature 5

From these assignments, the system can compute the translational and rotational offsets required to align the analytic object with this cluster of image features. Figure 18 shows the assignments of object features to image features and a dashed line indicating the location of the hinge implied by these assignments.

Occasionally, there is no unique "largest" completely connected subgraph. In that case, each subgraph is used to form a hypothesis. Thus, the analysis of a graph can produce more than one hypothesis. It is the responsibility of the next phase of the process to determine which, if any, of these hypotheses are valid.

Fig. 18. Local-feature identities and a hypothesized hinge location.



2.4. Hypothesis Verification

The current system uses two tests to verify hypotheses:

It looks at the image for other object features that are consistent with the hypothesis.
It checks the boundary of the hypothesized object.

As the program finds image features that match predicted object features, it adds them to a list of verified features (which adds strength to the initial hypothesis). It also uses the verified features to improve its estimate of the position and orientation of the object.

Given the refined estimate of an object's location, the program checks the boundary of the hypothesized object to determine whether it is consistent with the image. To do this, the system rotates and translates the analytic boundary and analyzes the contents of the image along the boundary. It takes samples perpendicularly to the boundary and checks for dark-to-light transitions (see Fig. 19). Light-to-dark transitions and all-light samples are negative evidence. All-dark samples are neutral.

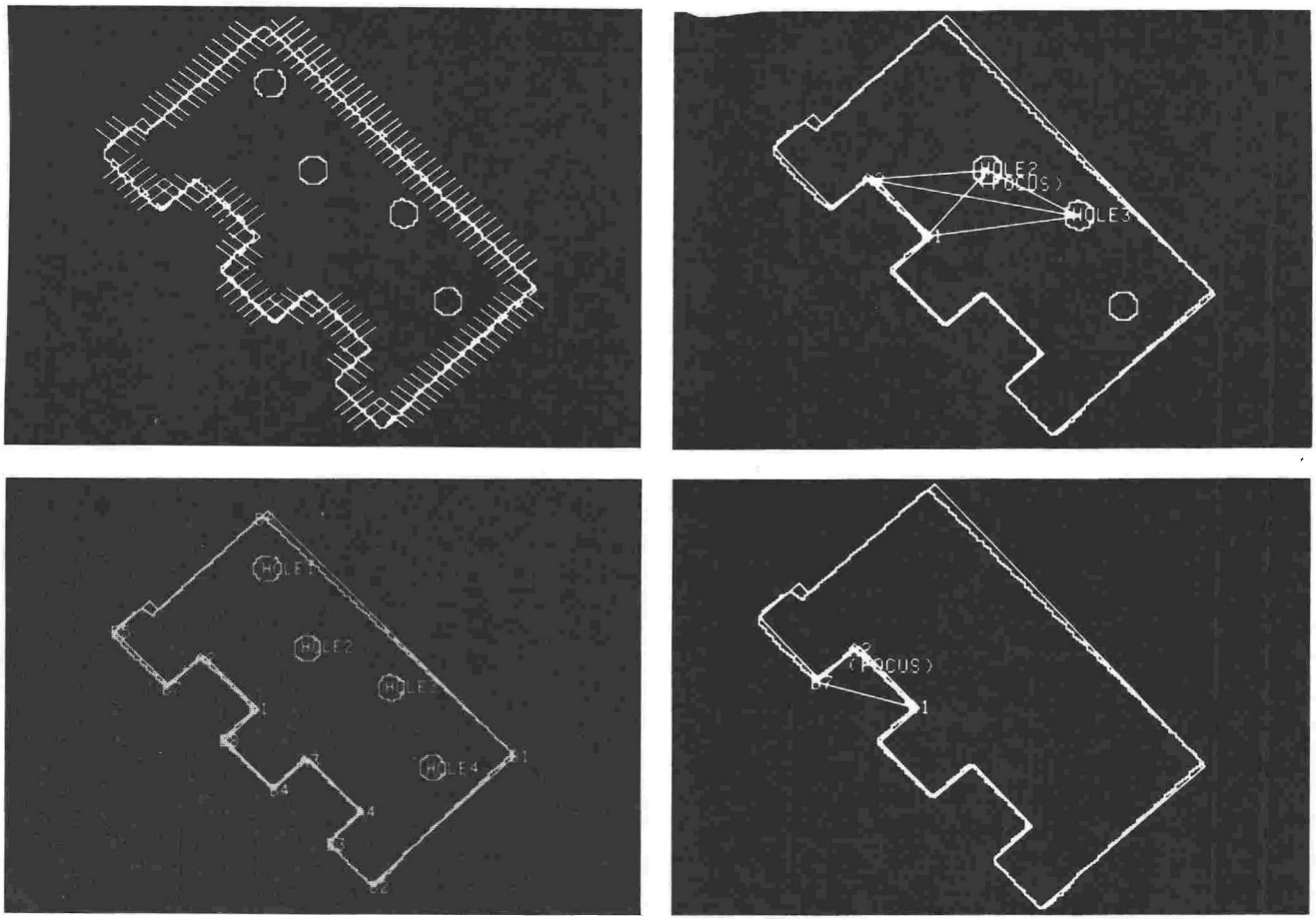
If a sufficient number of object features is located and the boundary verified, the system reports the identity of the object, its location, and its orientation. As information to be utilized in higher-level processing, it also reports how many features it was able to locate, the number it expected to see in the field of view, and the total number of features expected for the object. Finally, it marks the features

Fig. 19. Segments examined to verify hinge boundary.

Fig. 20. Verified hinge and its feature labels.

Fig. 21. Cluster of features used to locate the hinge when one hole is covered up.

Fig. 22. Cluster of features used to locate the hinge when all the holes are covered up.



as "explained" in the list of image features, so that additional processing does not attempt to reexplain them. Figure 20 shows the hinge with all of its features labeled.

When a cluster of features leads to two or more hypotheses, typically only one of them passes the verification tests. If two or more competing hypotheses pass the tests, however, the image is ambiguous and the system signals the problem by declaring each hypothesis as a "possible" match. Additional data are required to disambiguate such cases.

2.5. Examples and Extensions

To demonstrate the full capability of the run-time system, we will consider a sequence of increasingly difficult tasks. For example, if the hole that was used as the focus feature in the previous task is covered up, the system is forced to select a different focus feature—in this instance, another hole. A completely different group of nearby features is found (see Fig. 21) that matches a different cluster of object features. However, the implied hypothesis is the same.

If all the holes have been covered, the system is forced to use the next type of focus feature—in this case, an A-type corner. Once again, a completely different cluster of features is found (see Fig. 22). This cluster leads to two hypotheses, one for each of the rectangular intrusions into the part. The verification step selected the correct match.

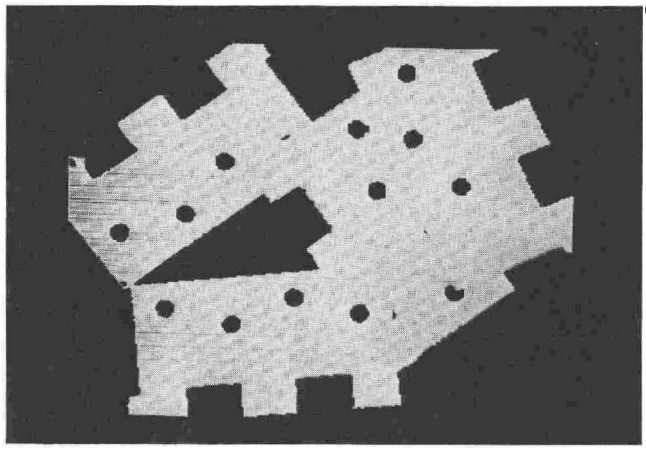
In a more complex scene (Fig. 23), the system uses many different clusters of local features to lo-

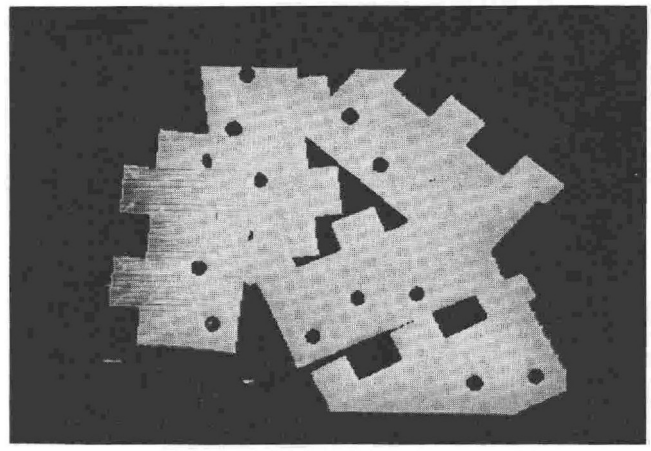
Fig. 23. Image of four overlapping hinge parts.

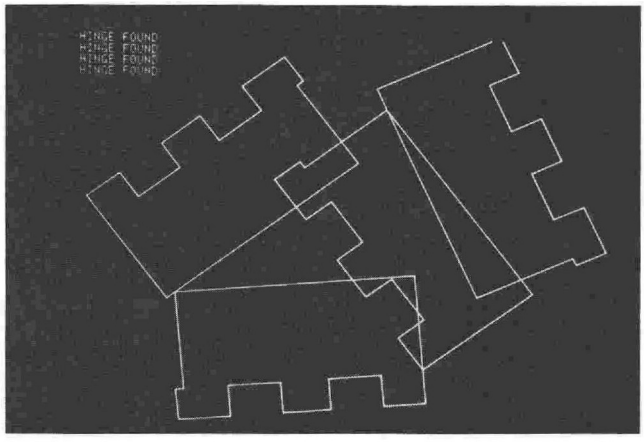
Fig. 24. Hinge parts located in Fig. 23.

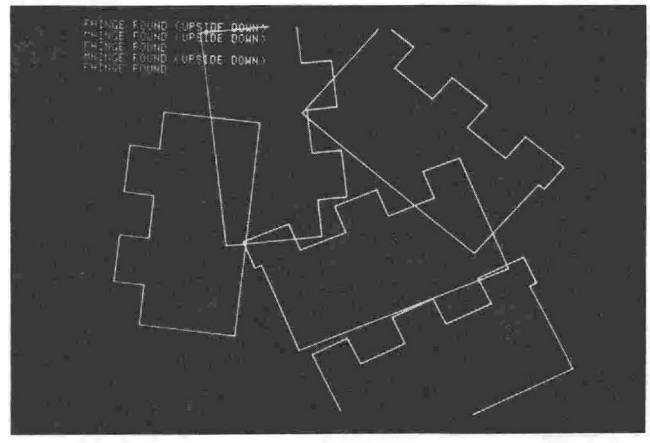
Fig. 25. Image of five hinges, some of which are upside down.

Fig. 26. Hinges located in Fig. 25.









cate the objects. This figure shows partially obscured hinges that are touching and overlapping. Using the techniques described above, the system is able to identify and locate all four hinges (see Fig. 24). With the current implementation, this processing takes approximately 8 s.

The run-time system also has the ability to recognize and locate objects that are upside down in the image. There is an operator-selectable option that invokes special decision-making procedures to cope with that possibility. If this option is selected, the run-time system does the following.

When nearby features are found, they are allowed two orientations; the one specified in the object file and its mirror image.

When the "mutually compatible" check is made, mirror-image orientations are allowed.

Once a hypothesis has been made, the system determines whether the object is upside down by analyzing three image features and their corresponding object features. Using any one of the three as a vertex, an included angle is computed for the image features and another one for the object features. If these included angles differ in sign, the object is upside down.

If the system determines that an object is upside down, it performs all subsequent processing, (i.e., feature verification and boundary checking) with a mirror image of the analytic object.

Figure 25 contains five hinges. The run-time system is able to identify and locate all five of them (see Fig. 26), even though three are upside down. This processing takes about 25 s in the current implementation. The main reason for the increase in pro-

cessing time for this example over that of the previous example (shown in Figs. 23 and 24) is that the system has to distinguish among four similar objects: the two-pronged hinge part, the three-pronged hinge part, and the mirror images of these parts. The additional possibilities increase the sizes of the graphs to be analyzed, which increase the processing times.

3. Training System

The primary design goal for the training system is to make it as easy as possible to use. There are basically two ways to do this. One is to develop automatic techniques for producing as much of the information as possible and the other is to "human-engineer" the system as well as possible. We have concentrated on the first approach because we are interested in developing ways to capitalize on the information that is commonly available for industrial tasks, such as descriptions of objects and constraints on viewing conditions. In this section, we will describe techniques for analyzing clusters of local features and show how they can be used to produce the information required by the LFF run-time system. Since these techniques include algorithms for building and analyzing representations of the geometry of local features, they could also be used to implement automatic training systems for other locational strategies based on local features (such as analyzing a histogram of the orientations suggested by pairs of local features [Stockman, Kopstein, and Benett 1982]).

The LFF training system is divided into two major parts: model acquisition and feature selection. The model-acquisition step produces models of objects to be recognized, while the feature-selection step chooses the best features for the LFF recognition procedure.

3.1. Model Acquisition

The purpose of the model-acquisition step is to construct models of the objects to be recognized. As stated earlier, these models include descriptions of the local-feature types and a list of local features associated with each object. There are several ways to construct these models. In traditional *teaching by showing*, a user shows the system several examples of an object and the system gathers statistics that are used to estimate the expected values of the object's global features and their variances. This approach is straightforward. It can be tedious, however, because a large number of examples is required to produce valid statistical models. Therefore, we are exploring alternative teaching methods that do not require multiple examples.

There are two steps in our approach to defining an object model. The first is specification of the nominal positions and appearances of the local features. The second is estimation of the variances associated with these positions and properties. In the current system, the user can either use a CAD model of an object to specify the nominal values or interactively point out features in an image of the object. The system then uses analytic and probabilistic models of the quantization errors to predict the variances about these values. The variances for a region's position and area are computed by means of formulas developed by Hill (1980). Variances for the other properties, such as a region's orientation or a corner's position, are estimated by a Monte Carlo technique that perturbs them in accordance with the predicted size of an individual pixel.

We have tuned the current system so that it is conservative in its estimates of variances. That is, any errors in its estimates are overestimates. Slightly inflated variances lead to somewhat longer recognition times, because more feature assignments are consequently made and analyzed. On the other hand, underestimates could cause the system to miss marginal objects. In any event, users can adapt the system to their specific needs.

3.2. Feature Selection

Given a set of object models, the feature-selection process chooses key features and key clusters of features for the LFF method. It selects a set of nearby features for each type of focus feature and then ranks the focus features according to the predicted costs of using them to recognize and locate objects.

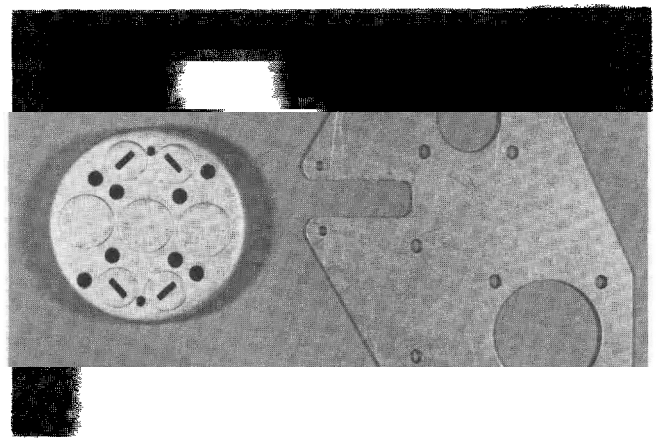


Fig. 27. A round electric box cover and a sheet-metal part.

To perform this selection and ranking, the process:

Identifies similar local features in different objects
Computes symmetries of the objects
Marks structurally equivalent features
Builds feature-centered descriptions of the objects
Selects nearby features
Ranks the focus features

Only the last two steps are specifically tailored to the LFF method. The other steps could serve in the preliminary analysis for other locational strategies.

3.2.1. Similar Local-Feature Types

Each object model contains descriptions of the types of local features associated with that object. The purpose of the first step in the feature-selection process is to determine the similarity of these features and produce a single list of local-feature types for the set of objects being considered. Identifying similar features is important, because the system needs to determine which features can be reliably distinguished on the basis of local information and which cannot. Or, put another way, the system needs to identify those features that can be distinguished only by the clusters of features near them. Consider the two objects in Fig. 27. Each contains three types of holes, and each contains eight-inch and quarter-inch round holes. Therefore, if the system is asked to recognize these two objects, it will combine the two lists of hole types into one list containing four types.

The identification of similar features is straightforward if the ranges of the property values describing the features are either disjoint or almost identical. The difficulty arises if several feature descriptions overlap in complex ways. The current system groups features together if the ranges of their property values (defined by the expected values and associated variances) overlap significantly. The user can specify the amount of overlap to be considered significant and can modify the groups suggested by the system. Reliance on the user for the final decision is a basic principle in our design of a semiautomatic programming system. The system automates as much of the process as possible and displays the results for the user's approval.

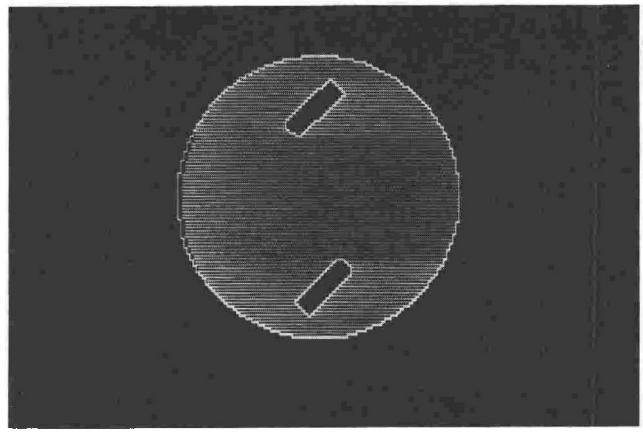
The incorrect grouping of features can lead to inefficient locational strategies. Grouping features that are dissimilar forces the run-time system to use more nearby features than necessary. The additional nearby features increase the sizes of the graphs to be analyzed, which in turn increase processing times. The source of the inefficiency is the use of nearby features to make distinctions that could be made reliably on the basis of the features' local appearances.

Not grouping features that are locally ambiguous can also lead to inefficient strategies. In this case, the feature-selection system is unaware of ambiguities to be resolved—and hence selects sets of nearby features that are not designed to make these distinctions. As a result, the run-time system is forced to proceed sequentially through possible interpretations of a focus feature until a verifiable hypothesis has been achieved. Since progressing through interpretations is time-consuming, it should be minimized. Thus, it is inefficient to form groups that are either too small or too large. Determining the best groups is difficult because it is a complex function of the cost of generating and verifying hypotheses. The current LFF training system forms reasonable groups for moderately difficult tasks and provides a convenient way for the user to experiment with variations of its suggestions for more difficult tasks.

3.2.2. Symmetries of Objects

Two fundamental properties of two-dimensional objects are their rotational and mirror symmetries.

Fig. 28. A twofold rotationally symmetric object.



These properties are important in the recognition and location of objects because they can be used to identify key features that determine the orientations of the objects and differentiate them from those that are similar. These properties are especially important in performing industrial vision tasks, in which symmetric, or almost symmetric, objects are common. For example, the round part in Fig. 27 is twofold rotationally symmetric (i.e., its appearance is unchanged by a rotation of 180°). If it occasionally occurs upside down in the scene, it is important to know that it is not mirror-symmetric. Since it is not mirror-symmetric, a vision system can detect upside-down parts.

A few artificial intelligence programs have performed symmetry analysis. Evans's program (1968), which worked geometric analogy problems, tested the primitive figures to see whether they were mirror-symmetric about a horizontal or vertical axis. Gips's program (1974) analyzed two three-dimensional strings of cubes to determine whether they were rotational or mirror transformations of each other. Perkins's program (1978) used a form of correlation to determine whether a pattern was rotationally symmetric. The latter program, like its counterparts, was not intended to perform a general-purpose symmetry analysis aimed at understanding local features, clusters of local features, and their properties. It is precisely this kind of comprehensive understanding of objects for which the analysis in this section are being developed.

Kanade (1981) has investigated skewed symmetry in images and its relation to the gradient space of surface orientations. He uses skewed symmetry to hypothesize real symmetry in a three-dimensional scene and to constrain the associated surfaces. Wechsler (1979) describes an algorithm that decomposes two-dimensional regions into mirror-symmetric components. His approach employs mathematical descriptions and tests for symmetry that are similar to the ones described in this section, but his goal is to describe regions rather than characterize clusters of local features. Silva (1981) describes a program that derives rotational symmetry axes for three-dimensional objects from multiple views of the objects. His task is more difficult than the problem discussed in this section because he is only given a set of images of the object, not an analytic model of it.

In the LFF system, local features are characterized by the following attributes:

Type (e.g., 90° corner or quarter-inch hole)
X-y position in the object's coordinate system
Orientation, if any, with respect to the x axis of the object's coordinate system
Rotational symmetry

Round holes do not have an inherent orientation. By convention, the orientation of a corner is the orientation of the bisector of the angle pointing into the object's interior. Rectangular holes, such as the ones in Fig. 28, are twofold rotationally symmetric and are assigned a nominal orientation along the major axis of the rectangle.

The position and orientation of a feature with respect to the centroid of the object are important quantities for symmetry analysis. They are defined in terms of the vector from the centroid of the object to the center of the feature. We refer to this vector as the *central vector* for the feature. The angular position of the feature is defined as the counterclockwise angle from the x-axis of the object to its central vector. The distance of the feature from the centroid is the magnitude of the central vector. The relative orientation of the feature is the smallest counterclockwise angle from the central vector to one of the feature's axes of symmetry (see Fig. 29). The relative position and orientation of one feature with respect to another is defined in terms of the three factors D, θ_1, and θ_2, as illustrated in Fig. 30.

Fig. 29. Diagram of the relative position and orientation of a feature with respect to an object.

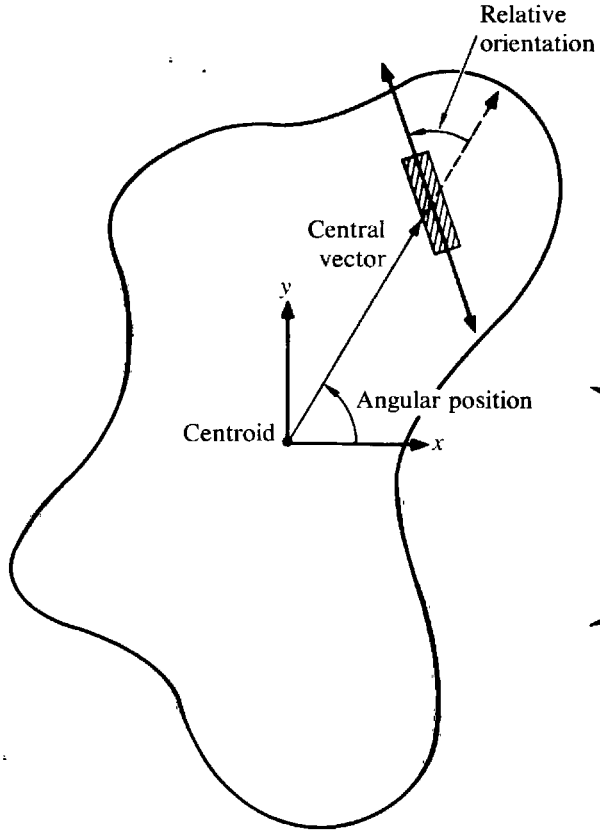


Fig. 30. Diagram of the relative position and orientations of one feature with respect to another.

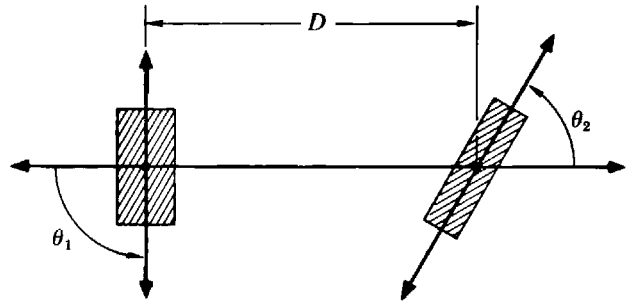
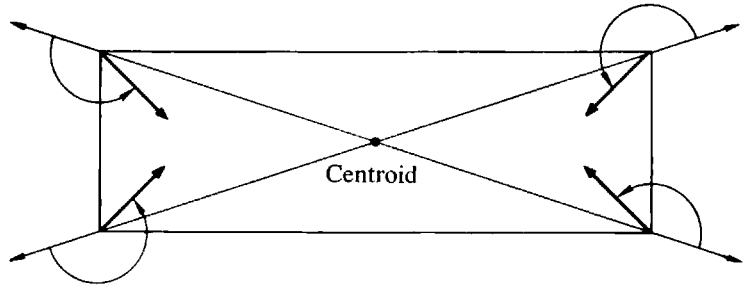


Fig. 31. Relative orientations of the corners of a rectangle.



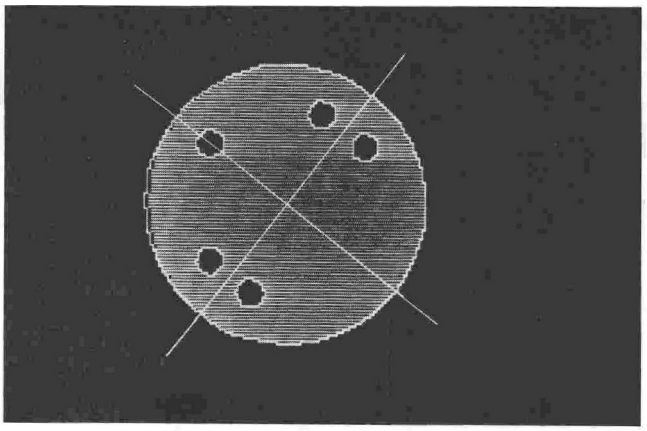
The LFF system determines the symmetries of a two-dimensional object in three steps: (1) formation of groups of similar features that are equidistant from the centroid of gravity of the object, (2) computation of the symmetries of the individual groups, and (3) computation of the object's symmetries with respect to those of the groups.

Features are designated as similar for the purposes of group formation if they are the same type, are equidistant from the object's centroid, and share the same orientation relative to their central vectors. Therefore, if two features are similar, the object can be rotated about its centroid so that one feature will be repositioned at the other feature's original position and orientation. For example, the four corners of a square are similar. However, only the diagonally opposite corners of a rectangle are similar, because the relative orientations of two adjacent corners are different (see Fig 31). Therefore, in forming groups for symmetry analysis, the system divides the four corners of a rectangle into two groups.

The second step in the symmetry analysis computes the rotational and mirror symmetries of each group. If a group of N features is D-fold rotationally symmetric, then D must be a divisor of N. Therefore, to determine the rotational symmetry of a group, it suffices to produce the list of factors of N and test the group for D-fold symmetry for each factor, the largest first. To test for D-fold symmetry, the system adds $360/D°$ to each of the angular positions of the features and compares the new positions with the original. If all the new positions are within a specified tolerance of the original ones, the group is D-fold rotationally symmetric.

If a set of two-dimensional points is mirror-symmetric, there is an interesting relationship between the axis of symmetry and the axis of least second moment: they are either identical or perpendicular to each other (Bolles 1979*b*). Therefore, to test a set of points for mirror symmetry, it is not necessary to do so in all possible orientations; it is necessary only to compute the axis of minimum second moment and to test the set for mirror symmetry about that axis and

Fig. 32. An object, its axis of least second moment, and the axis perpendicular to that axis.



the one perpendicular to it. Consider the five holes in Fig. 32, which are mirror-symmetric. The axis of second moment is the axis that points to the upper right corner. The pattern is not mirror-symmetric about the latter axis, but it is mirror-symmetric about the axis perpendicular to it.

For a group of features produced by the first step in the symmetry analysis to be mirror-symmetric, their centers must form a mirror-symmetric pattern, and each separate feature must be mirror-symmetric about its central vector. The pair of slots in Fig. 28 is not mirror-symmetric, because the individual features are not symmetric about their central vectors. To test a group for mirror symmetry, the system first checks one feature for mirror symmetry about its central vector and then checks the pattern of the centers for mirror symmetry.

It is interesting to note that a group of features, such as the group of five holes in Fig. 32, can be mirror-symmetric and not rotationally symmetric. It is also possible for a group of features to be rotationally symmetric, but not mirror-symmetric (e.g., consider the pair of slots in Fig. 28).

The third step of the symmetry analysis computes an object's symmetries with respect to those of the groups. The rotational symmetry of the object is easy to compute. It is the greatest common divisor of the symmetries of the groups (Bolles 1979*b*). The mirror symmetry and associated axes of mirror symmetry are more complicated to determine. Once the rotational and mirror symmetries of the groups are known, it is possible to compute a list of mirror-symmetry axes for each group. The basic idea is to construct these lists and intersect them to produce the mirror-symmetry axes of the object. However, there is one special case to be considered. This case is illustrated by the four corners of a rectangle. As stated above, they are divided into two groups according to their orientations relative to their central vectors. Neither group is mirror-symmetric, because the features are not mirro-symmetric about their central vectors. But the set of four corners, taken as a single group, is mirror-symmetric. The problem is that groups can occur in conjugate pairs. Therefore, to test an object for mirror symmetry, the system locates mirror-symmetric groups and pairs of groups that form mirror-symmetric patterns conjointly. If all the features of an object occur in mirror-symmetric patterns and there exists at least one common mirror-symmetry axis, the object is mirror-symmetric.

We have implemented these tests and confirmed that they work well within the LFF system. We also tried to incorporate them in a system for building models automatically, for a large class of two-dimensional objects, from images of the objects. We found this goal hard to achieve, as it was difficult to formulate a general definition of "significant feature."

3.2.3. Structurally Equivalent Features

If an object is rotationally symmetric, its features occur in groups whose members are structurally equivalent. That is, they cannot be distinguished on the basis of their local appearance or on the basis of the relative positions and orientations of other features in the object. For example, since the round object in Fig. 27 is twofold rotationally symmetric, its features occur in pairs whose members are structurally equivalent. The implication of structural equivalence for the LFF method is that only one member of a group of structurally equivalent features has to be analyzed. Therefore, if an object is rotationally symmetric, the number of features to be considered can be reduced by a factor equal to its rotational symmetry. Any reduction in the number of features is important because of the combinatorial aspects of the matching problem. A reduction in the number of features by a factor of two or more is quite significant.

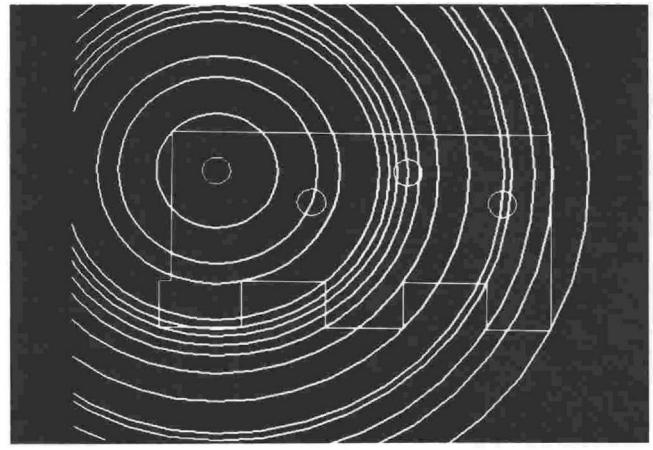


Fig. 33. Feature-centered description of the hinge with respect to one of its holes.

Having determined the symmetry of an object, the system marks as duplicates those features that are not to be regarded as focus features. Our initial marking procedure arbitrarily divided the features into rotational units, according to their angular positions with respect to the object's x-axis, and marked all features except those in the first unit. For example, for an object that was twofold rotationally symmetric, all features whose angular positions were between 180 and 360° were marked as duplicates. However, since the LFF system concentrates on local clusters of features, an improved marking procedure was developed that locates the most compact cluster of features constituting a complete rotational unit. For the round object in Fig. 27, this procedure divided the features into top and bottom halves, because the features are more closely grouped that way than in another kind of partition. This new strategy maximizes the intersection of the sets of nearby features (relative to a focus feature), which in turn minimizes the number of possible interpretations of the nearby features.

3.2.4. Feature-Centered Descriptions

The next step after the marking of duplicate features is to build feature-centered descriptions of the objects for each unique feature. These descriptions are designed to encode the information necessary for selecting nearby features. They are essentially the same as the descriptions generated by rotational symmetry analysis, except that they are centered on a local feature instead of the centroid of an object; in addition, the orientation of that feature, if any, is used in the formation of the groups.

The program builds a description for each nonduplicate feature. It does this by partitioning all features of the object, whether or not they have been marked as duplicates, into groups—and then computing the rotational symmetries of the groups. Features are grouped together if they are equidistant from the focus feature, are at the same orientation with respect to vectors from the focus feature, and if the focus feature is at the same orientation with respect to vectors from the features to be grouped (see Fig. 30 for these relative angles). If two features are grouped together, they cannot be distinguished by their relative positions or orientations with respect to the focus feature. Looked at another way, if two features are in the same group, the structural unit composed of the focus feature and one of those features is identical to the structure formed by the focus feature and the other nonfocus feature. Figure 33 illustrates the grouping of features around a hole in the hinge part.

3.2.5. Nearby-Feature Selection

Given feature-centered descriptions of the objects to be recognized, the nearby-feature selection step chooses a set of nearby features for each type of focus feature. It uses the feature-centered descriptions to suggest nearby features and evaluate them by locating matching features near other occurrences of the focus feature. As described in the discussion of the LFF task information (Section 2.1), a nearby feature is not really a feature; it is rather a set of criteria describing a class of features with respect to another feature. Therefore, in an image or in an object model there may be zero or more features that fit these criteria in the vicinity of a particular focus feature. The selection procedure creates descriptions of nearby features from examples of features in the models.

The basic requirement for a set of nearby features with respect to a focus feature is that the set contain enough features to identify the structurally different occurrences of the focus feature and establish the

Fig. 34. A unique cluster of nearby features for each hole.

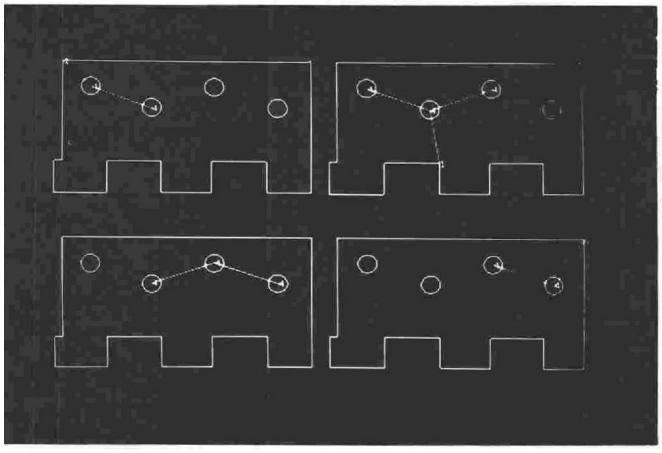
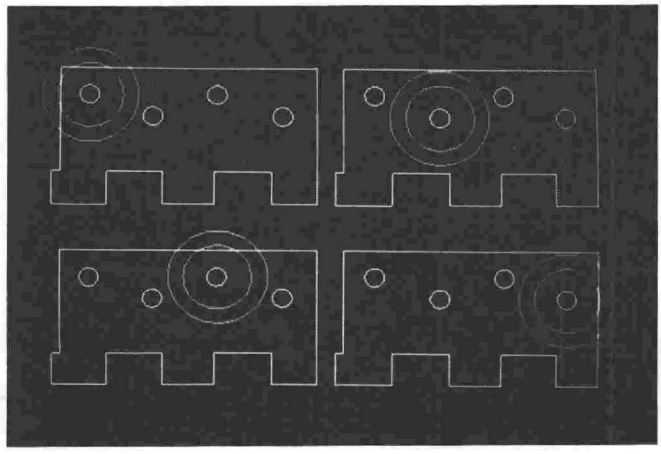


Fig. 35. Distance ranges for B-type corners with respect to a hole.



position and orientation of the object with respect to that feature. In other words, having found the features that match the nearby-feature descriptions, the system should be able to identify the focus feature and all nearby features, then compute the position and orientation of the object. For example, Fig. 34 shows four clusters of features that are sufficiently different to enable the hole they are centered upon to be identified and the object's position and orientation computed.

The current strategy for selecting nearby features is a two-step process:

1. Select nearby features to identify the focus feature.
2. Select additional features, if necessary, to determine the position and orientation of the object.

Features used to identify the focus feature are chosen first because the selected features almost always determine the position and orientation of the object as well. Features selected to determine the position and orientation often do not identify the focus feature.

When an additional feature is needed, those closest to the focus feature are considered first. The assumption is that features close to an observed feature are the most likely to be visible, both because they will probably be in the image and because they are less likely to be occluded.

To illustrate the selection process, consider the task of choosing nearby features for the hinge part shown in Fig. 14. There are three types of local features and hence three possible focus features: holes, A-type corners, and B-type corners. Let us start by selecting features for a hole-type focus feature. The first goal is to select nearby features that will allow us to identify which of the four holes has been found. To do this, the system selects features that differentiate between pairs of possible interpretations. That is, it chooses features to distinguish hole 1 from hole 2, hole 1 from hole 3, and so on. To distinguish hole 1 from hole 2, it selects corner B6 with respect to hole 1 (see Figs. 14 and 35). Corner B6 is the closest feature to either hole 1 or hole 2 and can be used to distinguish hole 1 from hole 2 because the latter does not have a B-type corner at that distance. In fact, since hole 1 is the only hole to have a B-type corner near it like B6, this one feature distinguishes hole 1 from any of the other three holes.

If the run-time system finds a hole and a B-type corner such that the corner is at B6's relative position and orientation, the hole is probably hole 1 and the corner is probably B6. For a couple of reasons, the system cannot be absolutely positive that the pair of features it has found is B6 next to hole 1. First, two hinges may be stacked in such a way that a B-type corner appears at the right distance and orientation with respect to another hole. Second, some noise along the boundary may look like a corner that just happens to be at the right distance and orientation with respect to a hole. Even though the system cannot be sure that the hole is hole 1 and the corner is B6, finding a pair like that is generally sufficient

Fig. 36. Sequence of nearby features automatically selected for hole-type focus features.

Goal	Selection	Additional Occurrences	Distinction 1×2	1×3	1×4	2×3	2×4	3×4	Orient. 1	2	3	4
Disting. 1×2	B6 wrt Hole1	—	*	*	*				*			
Disting. 2×3	A1 wrt Hole2	—	*			*	*			*		
Disting. 3×4	B1 wrt Hole4	—			*		*	*				*
Orientat. 3	Hole2 & Hole4 wrt Hole3	(H2 wrt H1 H1&H3 wrt H2 H3 wrt H4)	*	*			*	*	*	*	*	*
			3	2	2	1	3	2	2	2	1	2
ADDITIONS FOR A REDUNDANCY FACTOR OF 2												
Disting. 2×3	A3 wrt Hole3	A2 wrt Hole1	*		*	*		*	*		*	
			4	2	3	2	3	3	3	2	2	2
ADDITIONS FOR A REDUNDANCY FACTOR OF 3												
Disting. 1×3	B1 wrt Hole3	B2 wrt Hole4		*	*	*	*				*	*
Orientat. 2	A2 wrt Hole2	—	*			*	*			*		
			5	3	4	4	5	3	3	3	3	3

evidence for hypothesizing an occurrence of the object.

Having selected the B-type corner to distinguish hole 1 from hole 2, the program's next step is to evaluate the contributions that feature makes toward distinguishing other pairs and determining the object's orientation with respect to other holes. As already mentioned, the B6-type corner distinguishes hole 1 from hole 3 and hole 1 from hole 4. It can also be used to compute the orientation of the object relative to hole 1. Since the other holes do not have matching B-type corners, this nearby feature does not help determine the orientation of the object with respect to them.

The first line in Fig. 36 summarizes the contribution of this B-type corner near a hole. The goal of the selection procedure was to select a feature to distinguish hole 1 from hole 2 (written "Disting. 1 × 2" in Fig. 36). The corner B6 relative to hole 1 was selected. None of the other holes had matching B-type corners. Therefore, that feature distinguishes hole 1 from hole 2, hole 1 from hole 3, and hole 1 from hole 4. It also determines the orientation of the object with respect to hole 1. The remainder of Fig. 36 summarizes additional selections for hole-type focus features.

Since the B-type corner does not distinguish hole 2 from hole 3, the next subgoal of the feature-selection procedure is to choose a feature that will do this. It chooses A1 relative to hole 2 because it is the closest feature to either hole 2 or hole 3, whereas hole 3 does not have a matching A-type corner. As indicated in Fig. 36, none of the other holes have matching A-type corners. Corner A4 is at approximately the same distance from hole 4 as A1 is from hole 2, but its relative orientation is sufficiently different (as determined by the variances in the object model) to be distinguishable.

To differentiate between hole 3 and hole 4, the system selects B1 with respect to hole 4. After this nearby feature has been added to the list and the contributions updated, the only subgoal left to be achieved is to establish the orientation of the object with respect to hole 3. To satisfy this subgoal, the system selects the closest unused group of features near hole 3. Since none of the features near hole 3 have been used, the first group, which is a pair of holes, is added to the list. Hole 1 and hole 4 each have one matching hole nearby, while hole 2 has two such holes. Therefore, locating these holes can distinguish hole 1 from hole 2, hole 1 from hole 3, hole 2 from hole 4, and hole 3 from hole 4. Since patterns

of holes around all four focus features are rotationally asymmetric, they can be used to determine the orientation of the object with respect to their focus features.

This choice of holes near other holes points up a weakness in the current selection procedure. It is designed to use the closest unused feature that can achieve the goal. Otherwise, it could have selected A3 with respect to hole 3 to determine the orientation of hole 3. A3 is slightly farther from hole 3 than the pair of holes. A3 is a better choice, however, because none of the other holes have matching A-type corners, which means that fewer features would be required at run time to perform the task.

Figure 34 shows the clusters of features implied by the four types of nearby features that have been selected for a hole-type focus feature. These clusters are minimal in the sense that missing one of the features at run time could lead to an ambiguity. A feature might not be detected for several reasons. It might be occluded by another part; the part might be defective in such a way that it did not have a B-type corner in the expected position; the feature might be out of the camera's field of view; it might be so distorted that the feature detector could not recognize it. In the hinge example, missing B6 with respect to hole 1 results in an ambiguity. The system expects to find B6 and hole 2 near hole 1. If B6 is not detected, the pair of holes is ambiguous; it could consist of hole 1 and hole 3, hole 2 and hole 3, or hole 3 and hole 4.

If users of this automatic system want to incorporate some redundancy into the locational process, they can request the system to include enough features to achieve each goal in two or more ways. The numbers under the columns in Fig. 36 indicate the number of discrete ways of achieving each subgoal. These numbers are referred to as *redundancy factors*. The first row of numbers summarizes the contributions of the first four feature selections. There is at least one way to achieve each goal, two ways to achieve six of the goals, and three ways to achieve two of them. If the user wants at least two different ways to accomplish each goal, the system needs features that distinguish hole 2 from hole 3 and determine the orientation of the object with respect to hole 3. As indicated in Fig. 36, the system selects A3 with respect to hole 3, which happens to satisfy both subgoals. To obtain a redundancy factor of 3, the system adds two more types of nearby features.

The higher the redundancy factor, the lower the probability of not detecting enough features to identify the focus feature. However, increasing the list of nearby features leads to larger graphs to be analyzed, which in turn lengthens the processing time required to make hypotheses. In the case of the holes in the hinge part, the average graph sizes are 10.75, 11.75, and 13.00 nodes for redundancy factors of 1, 2, and 3 respectively. Sometimes the increase is more dramatic. For example, the graph sizes for A-type corners in the hinge are 5.50, 8.00, and 14.75 nodes for redundancy factors of 1, 2, and 3.

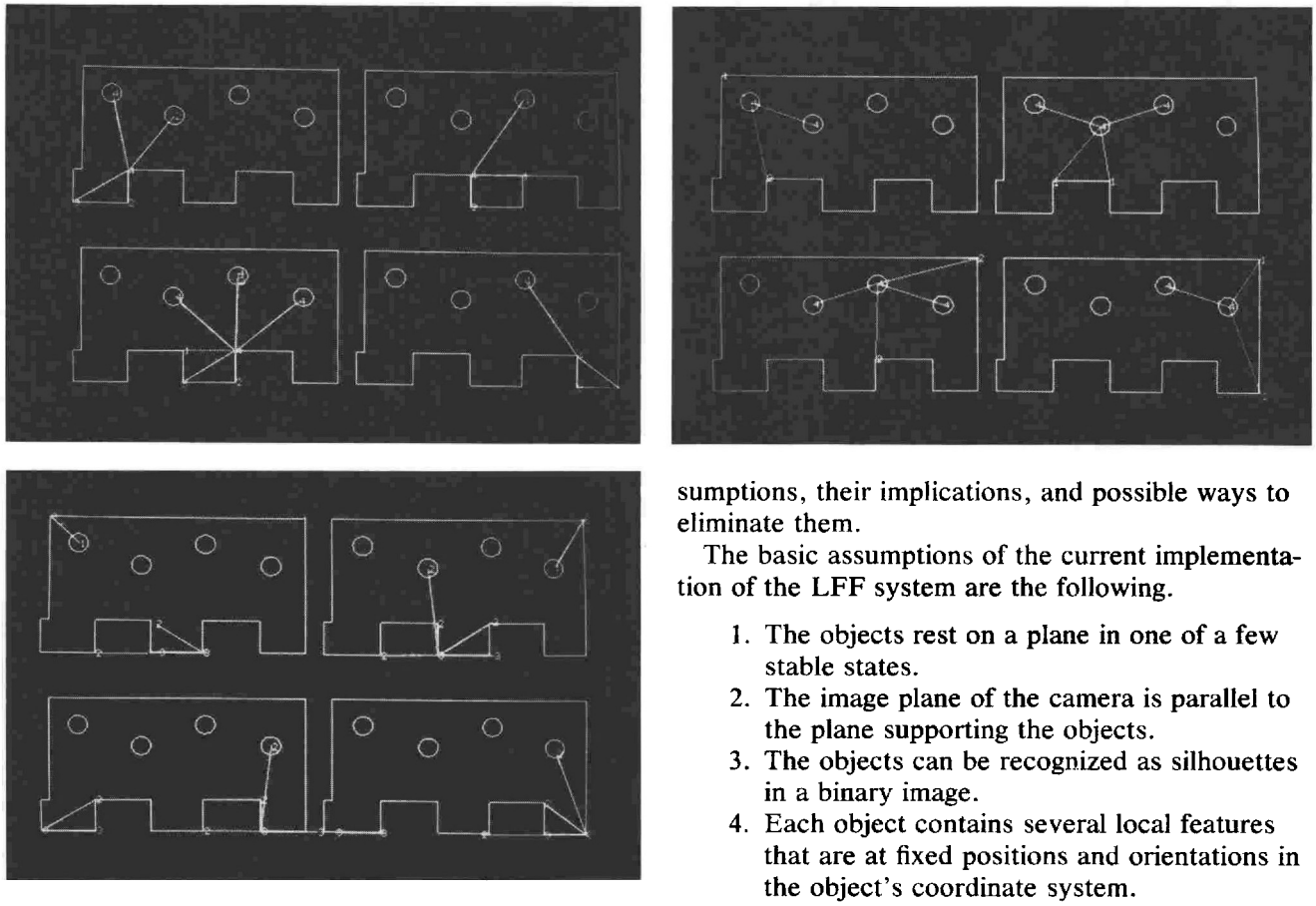
The optimum redundancy factor is a function of several variables, including the costs of building and analyzing graphs for producing hypotheses, the costs of verifying hypotheses, and the probability of missing features at run time. For the current implementation, we have found that redundancy factors of 2 or 3 work well for images containing four or five overlapping objects, such as the hinge part. The cost of making a hypothesis is approximately five times the cost of verifying one, but the probability of not detecting a feature at run time is so high that the extra features are needed to ensure finding a sufficient set.

After selecting nearby features for a hole focus feature, the system selects nearby features for the other two potential focus features, A-type corners and B-type corners. Figure 37 shows the clusters implied by the nearby-feature selections for all three focus features. A redundancy factor of 3 was used in these selections. These nearby features are the ones described in Fig. 9 and used for the examples in Section 2. In the Section 3.2.6, we will describe a procedure for deciding which focus feature to use first.

3.2.6. Focus-Feature Ranking

The LFF training system ranks the foci according to the sizes of the graphs they imply, since the construction and analysis of graphs are the most time-consuming steps in the locational process. The system constructs the clusters of features expected around each occurrence of a focus feature (as shown

Fig. 37. Nearby feature clusters having a redundancy factor of 3.



in Fig. 37), and then computes the sizes of the graphs from the lists of interpretations for each feature. The final ranking of a focus feature is based on the average size of its implied graphs. For the clusters shown in Fig. 37, the average graph sizes are 13.00 nodes for holes, 14.25 for A-type corners, and 14.50 for B-type corners.

4. Discussion

The LFF system can easily be trained to locate partially visible two-dimensional objects. Like all methods, however, it is based on a set of assumptions that implicitly define the class of tasks it can perform. In this section, we will discuss these assumptions, their implications, and possible ways to eliminate them.

The basic assumptions of the current implementation of the LFF system are the following.

1. The objects rest on a plane in one of a few stable states.
2. The image plane of the camera is parallel to the plane supporting the objects.
3. The objects can be recognized as silhouettes in a binary image.
4. Each object contains several local features that are at fixed positions and orientations in the object's coordinate system.
5. The objects can be distinguished on the basis of relatively small clusters of nearby features.
6. Speed and reliability are important.
7. The more the training phase can be automated, the better.

The first and second assumptions combine to restrict the tasks to those that can be performed by two-dimensional analysis. The assumptions imply a simple one-to-one correspondence between features in an image and features on an object. There is a direct correspondence between distances and orientations in the image and distances and orientations on the support plane. The effects of a perspective projection are minimized. Since three-dimensional distances and orientations can be measured directly in range data, an LFF-type recognition strategy could be used to locate three-dimensional objects in range data. We are exploring this possibility.

Fig. 38. Image of one hinge part almost directly on top of another.

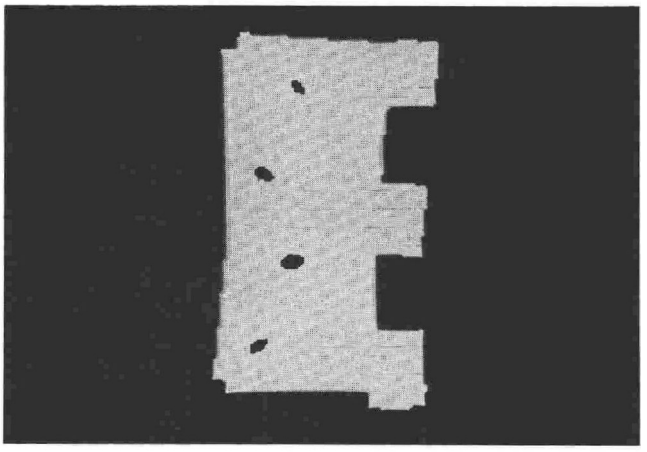
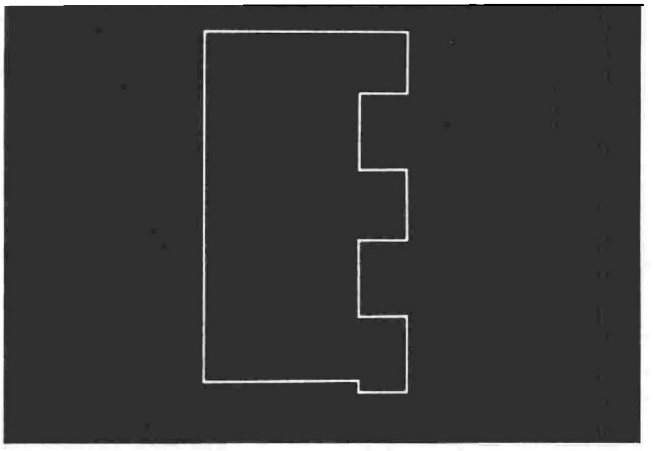


Fig. 39. Location of one of the hinge parts in Fig. 38.



The third assumption applies only to the current implementation. It is possible to implement an LFF system that uses gray-scale feature detectors. The types of object features and feature detectors would change, but the basic recognition strategy would remain the same. Such a system would be rendered particularly attractive by the inclusion of special-purpose hardware designed to locate corners and holes in gray-scale images.

The fourth assumption limits the class of objects that can be recognized to those that have several local features, such as corners and holes. The LFF system cannot locate objects such as French curves that are characterized by large, continuous arcs. Other methods, such as generalized Hough techniques (Ballard 1981) are better suited to this class of tasks.

The fourth assumption restricts the tasks to the recognition of rigid objects. We plan to explore ways to relax this constraint. In particular, we plan to investigate ways to locate objects with movable components. A simple strategy would be to ignore any features associated with such components. However, often these transient features contribute important, sometimes even crucial, constraints. How can they best be captured? The answer may lie in a multistage recognition procedure. We have already implemented the LFF system as a two-step procedure in order to deal with objects that are not mirror-symmetric. We originally treated an object and its mirror image as two separate objects. However, since the features and their relative distances are identical in both objects, we soon found that it was much more efficient to treat them as a single entity, relax our constraints on the relative orientations of features, and insert a second step in the recognition procedure that uses pairs or triplets of features to ascertain whether the object is right side up or upside down.

The fifth assumption emphasizes the fact that the LFF system is not designed to recognize an object by discerning one feature at one end of the part and another feature at the other end. It is designed to utilize local clusters of features. Figures 38–41 illustrate this point. Figure 38 shows two objects, one of which almost completely covers the second. The system recognizes one of these objects but not the second, because the detectable features are too far apart; they are not part of a local cluster. This concentration on local clusters is not as restrictive as one might think. For, if the objects are only slightly farther out of alignment (as in Fig. 40), local clusters emerge and the system is able to locate both of them (see Fig. 41). The LFF system, like all systems for locating partially visible objects, is better suited to tasks in which the objects are mostly visible, as opposed to tasks in which objects are almost completely occluded.

The LFF system was designed to locate industrial parts that may contain several identical features or patterns of features. However, as stated in the fifth assumption, the efficiency of the system depends on the fact that the objects can be distinguished on the basis of small clusters. Large clusters take longer to find. A corollary to this statement is that the LFF

Fig. 40. Image of two hinge parts that are not quite as closely aligned as the pair in Fig. 38.

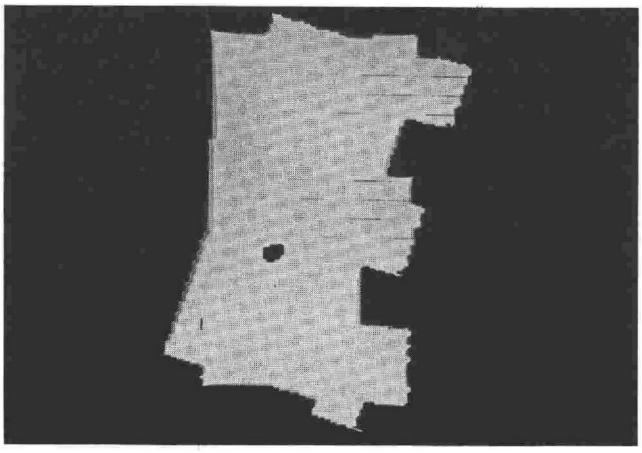
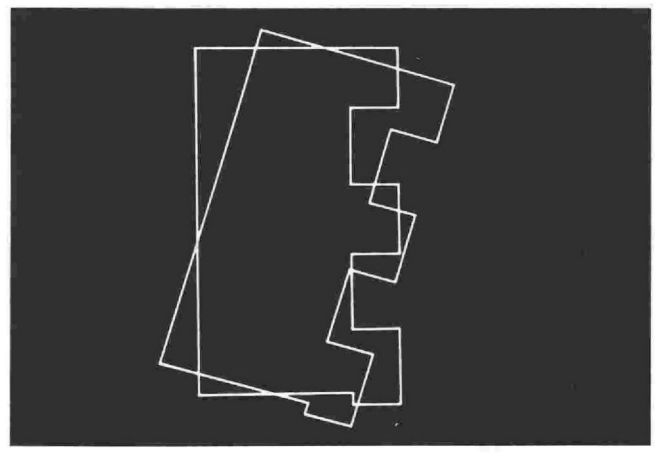


Fig. 41. Locations of the two hinge parts in Fig. 40.



system is slowed in its progress not by the number of objects to be recognized, but by the size of the ambiguous clusters. Fortunately, most parts—even machined parts that tend to contain patterns of identical features—have small distinguishing clusters.

The sixth assumption is somewhat of a catchall, but the intent is to rule out highly parallel matching techniques that require massively parallel hardware to make them practical. Such devices and methods will certainly be developed, but at least for the immediate future we are limited to a sequential machine. In light of that fact, we want to take advantage of the opportunity available in many industrial tasks to analyze models of the objects to be recognized, so as to improve the speed and reliability of the recognition process.

We plan to continue the investigation of general-purpose techniques for analyzing models of objects and selecting key features to be used at run time. We plan to extend the LFF selection procedure by having it explore more combinations of nearby features and by improving its evaluation of them. To this end, we are considering techniques for incorporating the cost and reliability of locating features, as well as other techniques for rating features according to the number of goals they help achieve and how they accomplish this.

The seventh assumption is not really an assumption. It is rather a statement of our philosophy. We believe that industrial vision systems will gain wide acceptance only when they can be easily trained. The easier the training, the more tasks they will be used for. The concept of selecting key features automatically is not limited to LFF strategy. A similar selection process could be used to select the best pairs of local features for a histogram-type matching scheme. Since the number of feature pairs increases as the square of the number of features, a judicious reduction in the number of features could generate a substantial saving.

In conclusion, it should be emphasized that the LFF system is more than just an efficient technique for recognizing and locating a large class of partially visible objects. It is also a semiautomatic programming system for selecting key features to be used in its own locational strategy. From a global perspective, however, it represents but one small step toward the development of fully automatic systems with the dual capability of both strategy and feature selection.

Appendix: Maximal-Clique Algorithm

For completeness, we describe here an algorithm for locating all maximal cliques (i.e., completely connected subgraphs) of a graph. The algorithm is essentially a restatement of one described by Johnston (1975). A more detailed description of this algorithm, its derivation, and its uses can be found elsewhere (Bolles 1979*a*).

The algorithm is a recursive procedure of three pa-