COMPUTER RECOGNITION OF OVERLAPPING PARTS USING A SINGLE CAMERA[1]

Parag Parikh[2]
Scott Berman[3]
C. S. George Lee[4]

April 1982

CENTER FOR ROBOTICS AND INTEGRATED MANUFACTURING

Robot Systems Division

COLLEGE OF ENGINEERING

THE UNIVERSITY OF MICHIGAN

ANN ARBOR, MICHIGAN 48109

---

[1]This project was supported by the Robot Systems Division of the Center for Robotics and Integrated Manufacturing (CRIM) at The University of Michigan, Ann Arbor, MI. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of The University of Michigan.

[2]Currently a senior in the Department of Electrical and Computer Engineering at The University of Michigan, Ann Arbor, MI.

[3]Currently a senior in the Department of Electrical and Computer Engineering at The University of Michigan, Ann Arbor, MI.

[4]Currently an Assistant Professor in the Department of Electrical and Computer Engineering at The University of Michigan, Ann Arbor, MI.

TABLE OF CONTENTS

ABSTRACT

A method to determine the position and the orientation of two overlapping parts using a single camera is described. The method is based on segmenting the image and labeling its various internal and external components. Detailed algorithms for automatic training, edge detection, picture segmentation and labeling, parts reconstruction, and parts recognition are provided. Results are summarized at successive stages. Advantages, limitations, and improvements are discussed.

1. Introduction

The use of computer vision to recognize overlapping objects and the use of computer-based robots to acquire, orient, and transport these objects will revolutionize manufacturing technology as we know it today. The bin-picking application, where robots are called upon to remove randomly scattered parts from a bin and place them on a conveyor with the proper orientation, will significantly improve and economize many manufacturing processes. Equipping the robot with "sight" and the ability to recognize overlapping objects is a first step in solving the bin-picking problem.

1.1. Literature Overview

In the last few years, many image processing techniques have been applied to the recognition of non-overlapping objects. The most successful of these use histogram techniques for finding local features such as holes, curves, and corners [8]; chain-encoding, gradient analysis, and Laplacian derivatives for locating edges and boundaries [9,10]; line-thining, line-merging, tracking, smoothing, and filtering techniques for image enhancement and sharpening [9,10]; and maximal cliques [4], template matching, edge-cues [11], and local feature cues [6] for recognizing the object, its position, and its orientation.

In fact, one system that not only determines the position and orientation of non-overlapping objects, but also passes that information to a PUMA robot for acquisition and transportation, has already been developed at The University of Michigan's Robotics Research Laboratory [12]. The basic method employed by this system is to compare previously taught parameters such as the area, the perimeter, the ratio of perimeter squared divided by area, etc. and a radial template with similar parameters calculated in real time.

Although this system and others like it, have been successful in the recognition of non-overlapping parts, they show severe limitations when attempting to recognize overlapping parts. The major problem is in distinguishing the image as two overlapping objects rather than one object, i.e. segmenting the image.

Nevertheless, some success in solving the bin-picking problem has been achieved. A group at the University of Rhode Island has developed a three-step method to acquire and orient overlapping parts in a bin [7,8]. First, using grey-scale computer vision techniques, they approximate the location of any one object. Second, using sensory feedback from a vacuum gripper, they make the robot pick up the object and transport it to a "orientation bin". Finally, using a non-overlapping objects algorithm, they orient the part and transport it to its final destination. Notice, however, that the algorithm will not necessarily acquire the topmost object. In fact, the initial acquisition by the robot is completely at random. While this may be sufficient for some applications, other applications require the recognition of overlapping objects in their overlapping state.

1.2. Overview of the Problem

The objective of this research project is to develop an algorithm using grey-scale computer vision techniques to recognize overlapping objects in their overlapping state. More specifically, the task is to detect the outside boundaries of the objects, construct both the overlapping edges and the hidden edges, and determine the position and the orientation of the top object and then the bottom object. The purpose of this report is to present an algorithm that accomplishes this task for two specific overlapping parts (see Appendix A). But more importantly, the purpose is to clarify and describe the various techniques used so that they may aid future research projects on this topic.

The particular techniques examined include: 1) training 2) image acquisition 3) edge detection 4) labeling 5) feature recognition 6) template matching and 7) objects recognition and reconstruction.

2. Problem Definition and Formulation

The task of determining the position and orientation of overlapping objects was divided into two phases. The first phase involved training the computer to recognize each object in its non-overlapping state. The second phase involved the actual recognition and reconstruction of the objects in their overlapping state. This second phase was divided into six stages as follows:

(1) image acquisition
(2) edge detection
(3) labeling
(4) feature recognition
(5) template matching
(6) objects recognition and reconstruction

The remainder of this section discusses the two phases and elaborates on the purpose, the function, and the methods involved. Detailed algorithms are omitted here for simplicity, but provided in the next section for completeness. Figure 1 shows the overall flow of the vision system.

2.1. PHASE 1 - Training

The first phase in the system is to train the computer and teach it the characteristics of each individual object in all of its non-overlapping stable states. This is an absolute necessity if the computer is to recognize and reconstruct the objects in an overlapping state when only partial information is available. Essentially, this phase is nothing more than automatically creating a "model" of the object. It should contain information about the set of local features that best describes the object. The procedure is to run a modified version of the same

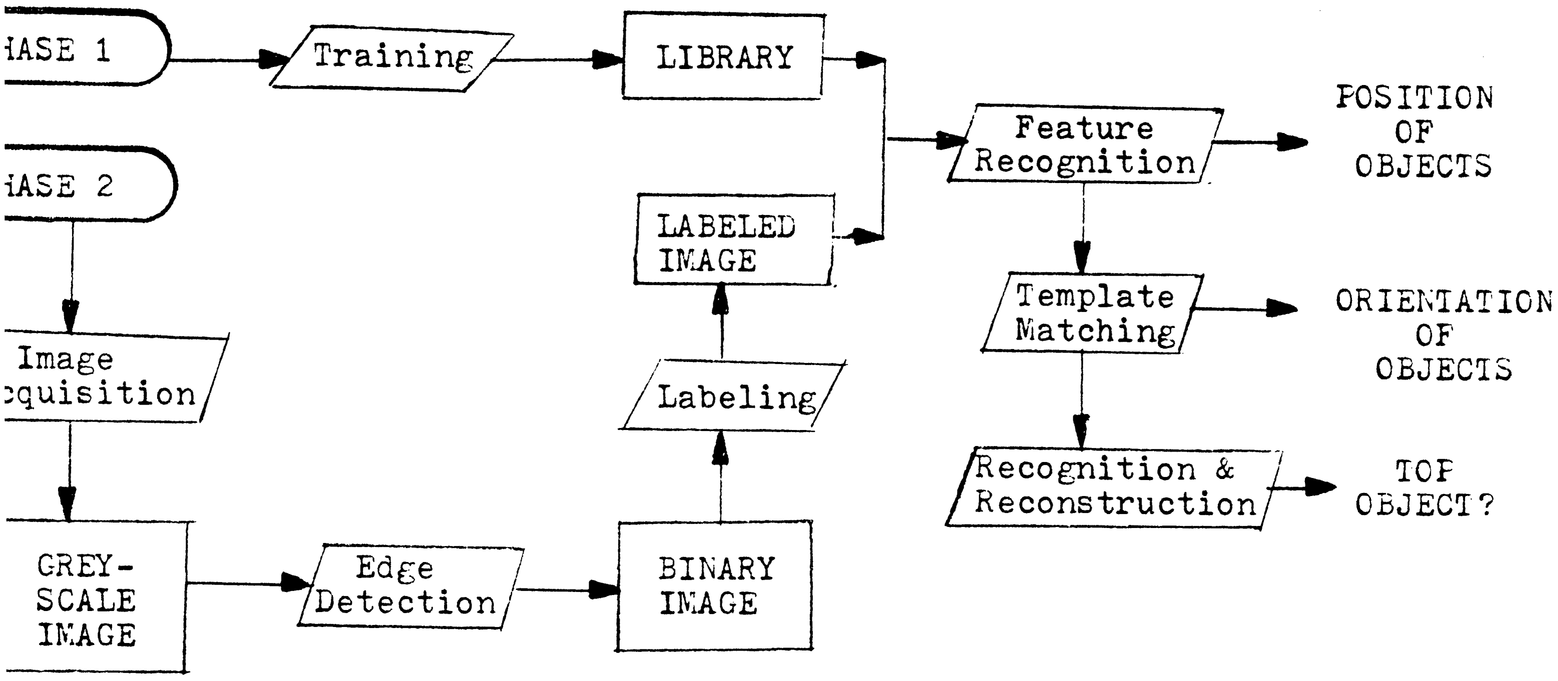


Figure 1 Overall Flow of the Vision System

algorithm to be used on overlapping objects, but instead of recognizing local features, such as holes, and trying to look them up in a library, it creates the library by storing parameters like area, perimeter, length of the horizontal axis, and length of the vertical axis of any local features that are present.

The training performed in this project concentrated primarily on the large circular hole in the middle of the object (see Appendix A). It calculates what can be called a "radial template" by starting from the center of the hole and computing distances to the outer edge for every one degree increment. The purpose of this template is not only to recognize the objects and determine which one is on top, but also to reconstruct any hidden edges and verify the recognition.

In any event, the thing to note is that the training needs to be done only once for each stable state of the objects used. This is a major step in automating the system.

2.2. PHASE 2 - Object Recognition

2.2.1. Image Acquisition Stage

The images are acquired using a single Hamamatsu T.V. Camera and run through a high-speed A/D converter as described in Appendix B. For development, the resolution of each picture was standardized to 256x256x8 bits.

2.2.2. Edge Detection Stage

The second stage in recognizing overlapping objects using computer vision is to detect not only the outermost edges but also any overlapping ones. After discussing various possibilities, an algorithm based on the gradient magnitude calculation was finally implemented.

The most successful version of the algorithm involves three major steps. The first step is the use of a gradient calculation to obtain a rough estimate of the location of the edges. The second step involves distinguishing between the actual edges and random noise, and eliminating the latter. And the third step involves the creation of a bounding rectangle or a window around the objects to facilitate further processing.

2.2.2.1. Gradient Calculation

The first step in the edge detection stage is to determine the location of all possible edges using a gradient magnitude calculation. Essentially, this involves comparing a small group of neighboring pixels around a central pixel and looking for sizable grey-scale intensity differences. If such differences exist, then that particular central pixel is flagged as a possible edge.

Two different gradient algorithms were tested. The first one uses a 3x3 window while the second one uses a 5x5 window. The 3x3 window algorithm yielded much thinner edges, and was chosen for further processing.

The basic calculation is as follows:

$$gradient = \sqrt{\Delta x^2 + \Delta y^2}$$

where:
Δx = grey-scale intensity difference in horizontal direction
Δy = grey-scale intensity difference in vertical direction

By thresholding the gradient value, most edge pixels can be detected and the image converted from grey-scale to binary. However, considerable non-edge pixels or "noise" pixels are also included. See Appendix D for images resulting from this step in the process.

2.2.2.2. Noise Elimination

The second step in the edge detection stage is to eliminate any random noise that was generated in the gradient calculation step. This is done by simply examining the neighboring pixels around every edge pixel and turning "off" (making a logical 0) those edge pixels that have less than a threshold number of pixels "on" (a logical 1) around them.

2.2.2.3. Border Creation

The third and last step in the edge detection stage is to create a bounding rectangle or window around the overlapping parts. Essentially, this involves taking the results of the noise elimination step and calculating the minima and maxima of the x and y coordinates where edge pixels are located.

The reason for drawing this window is to reduce the number of pixels that will need subsequent processing. Instead of processing all 256×256 pixels, only pixels that lie within the bounding rectangle or window need to be considered further. No time needs to be wasted in processing areas where the objects do not exist. See Appendix E for final results of the edge detection stage.

2.2.3. Labeling Stage

The third stage in the Object Recognition phase is the labeling technique. This is essentially a modified and enhanced version of the "Run Length Coding" algorithm described by Gerald J. Agin [2].

The major function of this stage in the recognition process is to separate the image into its individual components by labeling each with a different integer called a component number. For example, the leftmost image in Figure 2 is labeled to produce the rightmost image in Figure 2, where integer (component number) "0" is the background, "1" is the object, and "2" is the internal hole.

The advantage of this algorithm is that only one row in the image needs to be buffered and processed at any one time. All of the information gathered from the processing of previous rows is stored in a relatively small data structure. Also, all further processing can now be restricted to a subset of the original image or those pixels that are in the window created in the previous stage.

The general method is quite simple. First, the row to be processed is read from the binary image resulting from the edge detection stage (Appendix E) and scanned to determine the columns where 1 to 0 or 0 to 1 transitions occur. The distance between any two consecutive transitions is called a segment. For example, a row like in Figure 3 would produce transitions in columns 8, and 13, and would result in 3 segments as shown.

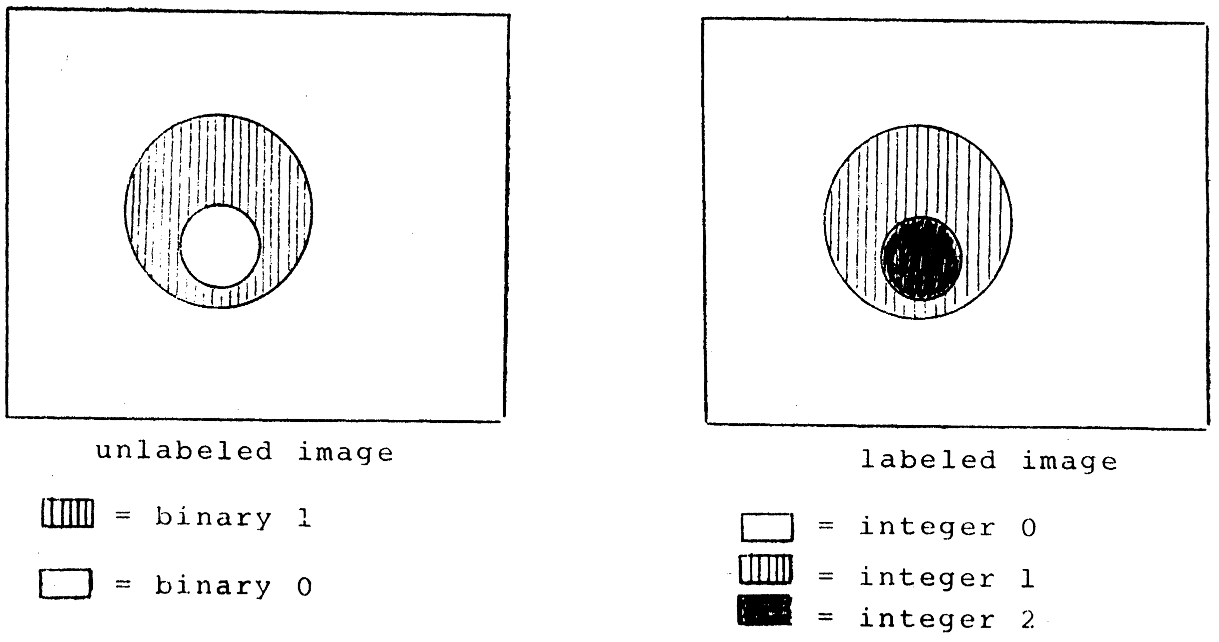


Figure 2 Labeling Technique

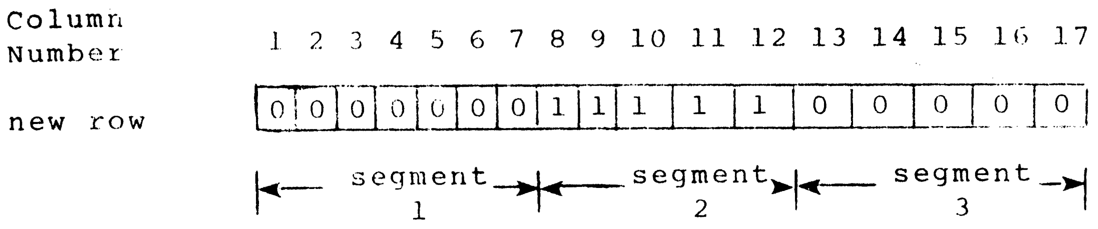


Figure 3 Determining Segments in a New Row

Second, each segment in the new row or the row being processed, called the <u>new segment,</u> is processed one at a time to determine how it fits relative to segments processed in previous rows. The basic idea is to perform "connectivity analysis" on each new segment by comparing it to the corresponding segment in the data structure. This corresponding segment is called the <u>current segment.</u> In performing this comparison, three possible cases can arise as shown in Figure 4. Case 1 is where the new segment is completely to the right of the current segment. Case 2 is where the new segment is completely to the left of the current segment. And Case 3 is where the new segment overlaps the current segment.

Case 1 occurs when there are more entries in the data structure than actually needed. Thus, one entry can be deleted and adjacent component numbers merged. Figure 5 shows that the third segment in the new row (the new segment), [XXXX], is a Case 1 when compared to the third segment in the old row (the current segment in the data structure). Thus, the component number "0" can be deleted from the data structure and the numbers "1" and "2" can be merged. The merging is necessary to combine component numbers that are connected and really part of the same "piece" of the image.

```
Current Segment:                 xxxx

New Segment (Case 1):                    xxxx

New Segment (Case 2):   xxxx

New Segment (Case 3):              xxxx
```

Figure 4 Three Cases Found in the Labeling Technique

```
Old Row:    0 2 2 0 0 1 1 1 1 0 0          0 2 2 0 0 2 2 2 2 0 0
New Row:    0 0 2 2 2 2 2[X X X X]         0 0 2 2 2 2 2
                 Before                          After
```

Figure 5 Processing Case 1

Case 2 occurs when there are less entries in the data structure than necessary. Thus, one segment must be added to the data structure and a new component number created. Figure 6 shows that the second segment in the new row (the new segment), [XX], is a Case 2 when compared to the second segment in the old row (the current segment in the data structure.). Thus, the new segment is added to the data structure and assigned the first available component number. In Figure 6, for example, the new segment is assigned the component number "2".

```
Old Row:    0 0 0 0 0 1 1 1 1 0 0          0 0 0 0 0 1 1 1 1 0 0
New Row:    0 0[X X]                       0 0 2 2
                 Before                          After
```

Figure 6 Processing Case 2

Case 3 occurs when the entries in the data structure are consistent with the number of new segments. Nothing is added to or deleted from the data structure. The new segment, in this case, is labeled with the same component number as its corresponding segment in the data structure. Figure 7 illustrates this case. The second segment in the new row (the new segment) is labeled with the same component number as the second segment in the old row (the current segment in the data structure).

Notice, however, that of all three cases only Cases 2 and 3 actually process the new segment. Case 1 simply keeps "cleaning up" the data structure until either a Case 2 or 3 match can be found to label that segment.

This fact is important when we consider what happens if the new row contains less segments than the old row (the data structure). In this case, all the new segments from the new row may be exhausted before the data structure can be "repaired". This is shown in Figure 8 where three Case 3's are processed, yet two segments remain unused in the data structure (segments labeled "3" and "0"). The problem with this is that it results in a portion of the background being labeled with a non-background component number (component number "2"). In this case, now called Case 4, repeated calls must be made to Case 1 (with merging

```
Old Row:   0 0 0 1 1 1 1 0 0 0        0 0 0 1 1 1 1 0 0 0
New Row:   0 0[X X X]                 0 0 1 1 1
                Before                        After
```

Figure 7 Processing Case 3

suppressed) until the number of segments in the new row agrees with the number of segments in the old row. Furthermore, the last segment in the new row must be forced to the background, component number "0", no matter what the data structure says. This was one of the major modifications made to Agin's "Run Length Coding" algorithm [2].

Another problem that arose involved the background number. Ideally, the background number should be labeled with the component number "0". However, because of the merging that occurs in Case 1, the component number "0" may be merged into another component number and lost. This is quite easily corrected by keeping track of what the background has been changed to, and changing it back to "0" at the end of the labeling algorithm. However, we must make sure that the component number "0" does not get used to label any other part of the image.

The labeling technique is really the heart of the overlapping parts recognition system. By separating individual components of the image, many local features can easily be recognized, areas and perimeters can be calculated, background and objects can be separated, and internal features such as holes of various sizes and shapes can be recognized and distinguished from each other. The labeling technique shows a great deal of potential in computer vision applications. See

Old Row:	0 0 1 1 2 2 3 3 0 0	0 0 1 1 0 0 3 3 0 0
New Row:	0 1 1 2 2 2 2 2 2 2	0 1 1 0 0 0 0 0 0 0
	Before	After

Figure 8 Processing Case 4

Appendix F for the various internal components separated by this algorithm for one particular overlapping objects image.

2.2.4. Feature Recognition Stage

The fourth stage in the recognition of overlapping objects is the recognition of any local features that were either modeled during the Training phase or acquired from an inspection of the objects.

At this point in the system, the overlapping objects image has been processed in two ways. First, it has been passed through the edge detection stage where edge pixels were found and separated from extraneous noise. Second, it has been passed through the labeling technique where various components have been labeled with different component numbers.

The function of this stage in the system, the feature recognition stage, is to scan the labeled image trying to recognize specific local features. The local feature that this particular system focuses on is the relatively large hole in the center of the objects used (see Appendix A). The reason that this local feature was selected is because it is the most apparent and easily distinguishable characteristic of the objects.

The basic method used to find and recognize the internal hole consists of three independent tests: 1) the area test 2) the ratio test and 3) the continuity test.

The first test in the sequence, the area test, is used to eliminate any component numbers that are either too large or too small to be the hole. This is done by comparing the area (number of pixels) occupied by each successive component number with the actual area of the hole. Those component numbers that do not have an area within a certain tolerance, perhaps 10-20% of the actual area, can safely be eliminated from further consideration.

The second test in the sequence, the ratio test, is used to distinguish holes of various shapes that might have the same area. The basic idea here is to compute the ratio of the number of pixels spanned on the major axis divided by the number of pixels spanned on the minor axis. For a circular hole, the ratio will be very close to 1. For an elliptical hole, the ratio will always be greater than 1. Of course, the more elliptical the hole, the greater the ratio. For purposes of this project, however, the ratio computed was the number of pixels spanned in the horizontal direction divided by the number of pixels spanned in the vertical direction. This deviation was prompted by the fact that the objects used have only one hole and it is perfectly circular.

The third and last test in the sequence is the continuity test. The purpose of this test is to separate the outer boundary of the hole from the hole itself (see Figure 9). The basic method of performing this test is to scan the labeled image along the horizontal axis looking for transitions from the component number being tested to some other component number or vice versa. If more than two such transitions are found, the component number being tested is not continuous and can be eliminated from further consideration as a hole.

In the recognition of two objects in their overlapping state, four possible conditions arise as a consequence of these tests: 1) no hole may be found (a partial hole is considered to be a no hole condition) 2) one hole may be found 3) both holes may be found or 4) more than two holes may be found. If, no holes are found, the system as it stands now will come to the unfortunate conclusion that the objects cannot be recognized. In this case, other local features must be selected to perform the recognition. For example, it might be feasible to search for a sharp corner or a smooth edge. However, if one, two, or more holes are found, the center of each hole can be computed and passed to the next stage in the system.

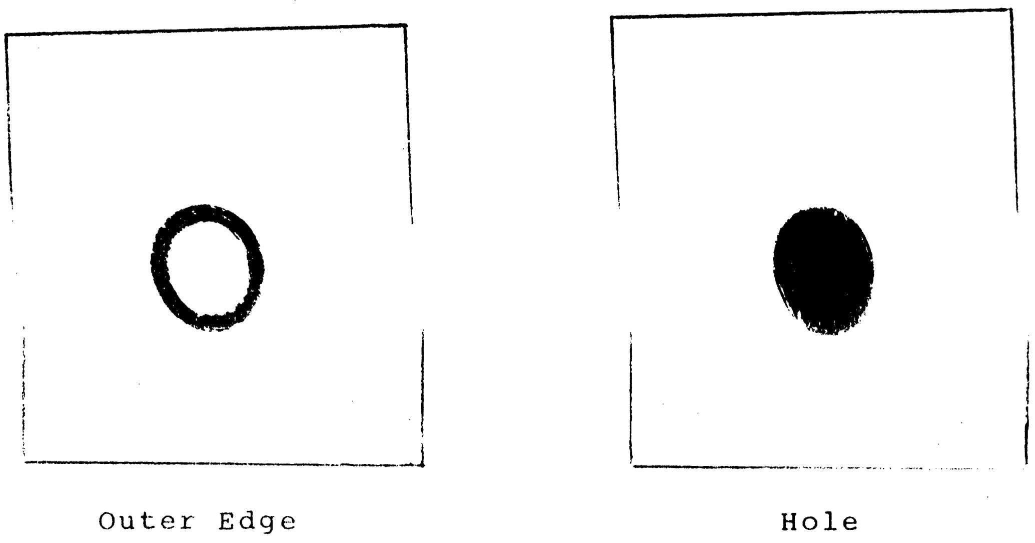


Figure 9 Distinguishing A Hole From Its Outer Edge

2.2.5. Template Matching Stage

The fifth stage in the recognition of overlapping objects is to compare the radial template(s) calculated in the Training phase with the radial template(s) calculated in real time from holes found in the previous stage, until a "best fit" match is found. This "best fit" match essentially yields the orientation of the object relative to the trained object. Once the orientation is determined, the next stage of the system can reconstruct any hidden edges and determine which object is on top.

The first step in the template matching algorithm is to calculate the radial template for each hole found in the previous stage. The method is exactly similar to the Training phase. The algorithm is to start from the center of the hole and compute distances to the outer edge for every one degree increment. The result is an array with 360 entries, one entry for each degree.

The second step in the algorithm is the actual comparison. The basic approach is to match one of the templates (arrays) calculated in real time with each template from the Training phase. By holding one fixed, and reindexing the other without destroying its internal order, the best match can be found.

The method is to match up the arrays (templates) so that the i^{th} element of one corresponds to the i^{th} element of the other, and performing a calculation to determine the total deviation as follows:

$$total = \sum_{i=0}^{359} \left(template1[i] - template2[i] \right)^2$$

for all i such that:

$$(0.5) \times template2[i] < template1[i] < (1.5) \times template2[i]$$

where:
$template1[i]$ = i^{th} element of the real time template
$template2[i]$ = i^{th} element of the trained template

The reason that only certain values of i are used is that when the magnitude of the difference between corresponding template values is very large relative to the template values, either one of those template values has crossed an overlapping edge or is in some way not representative of the object. For example, the dotted line in Figure 10 shows that at 180 degrees the template value has crossed the overlapping edge due to a gap in the overlapping boundary and no longer reflects its own template. Consequently, it must be omitted from the calculation.

After this first calculation is performed, one template is held fixed and the other template is shifted by one degree. Thus, the 1^{st} element in one corresponds to the 360^{th} element in the other, the 2^{nd} element in one corresponds to the 1^{st} element in the other, etc.(see Figure 11). Then, the total deviation must again be calculated exactly as before. In fact, this process must continue until the

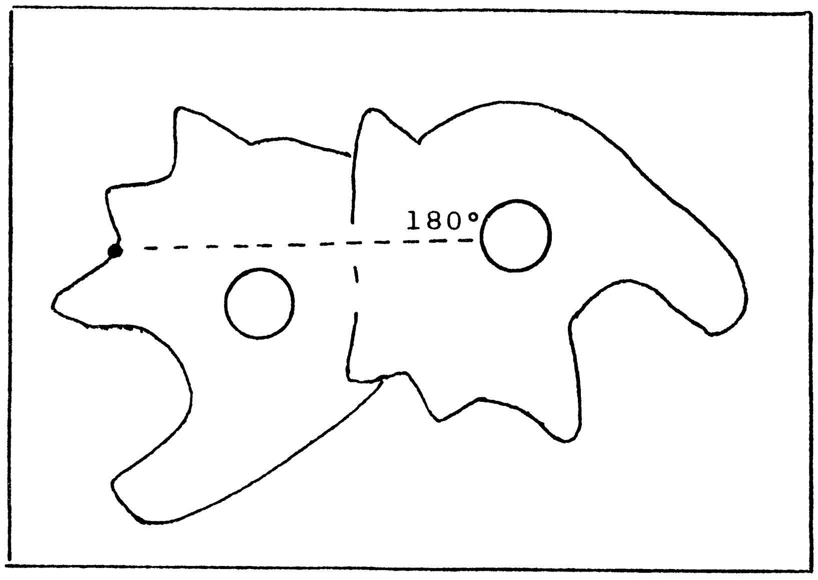


Figure 10 Erroneous Template Values

templates have been compared in all 360 possible shifted combinations and 360 totals are obtained. The template calculated in real time can now be compared to another template from the Training phase, representing another possible stable state of the object. The configuration that results in the smallest total deviation after all the trained templates have been considered must be the orientation of that particular object. Thus, the orientation can be represented by the stable state of the trained template used and the number of shifts that resulted in the smallest total deviation. If more than one hole is visible, the entire algorithm just described for template matching can be repeated to determine the orientation of each corresponding object.

This information can now be passed to the last stage in the overlapping objects recognition system to determine which object is on top and reconstruct

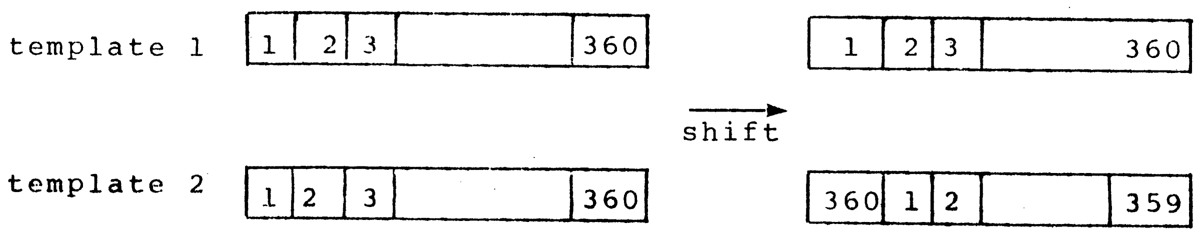


Figure 11 Shifting Template Before Calculations

any hidden edges.

2.2.6. Objects Recognition and Reconstruction Stage

The sixth and last stage in the recognition of overlapping objects is the determination of which object is on top and the reconstruction of any hidden edges.

Thus far, two key parameters have been computed. First, the feature recognition stage has resulted in the position (x and y coordinates) of each non-overlapped hole within the image frame. Second, the template matching stage has resulted in the best estimate of the orientation of the objects.

Without any further calculation, the objects can easily be reconstructed by simply redrawing the radial template that was trained in the first phase of the system at the correct position in the correct orientation. See Appendix G for examples of reconstructed images.

The process of determining which object is on top, however, is slightly more complex. The basic algorithm involves comparing the images obtained from the edge detection stage (Appendix D), with the radial template, after it has been properly positioned and oriented, for every hole and determining which one correlates or matches up the "best". This is the topmost object.

The basic procedure for correlating is to determine if an edge pixel lies in the edge detected image within a certain range ($\pm$ 3 pixels) of where the repositioned and reoriented trained template says it should lie. After this is done for all 360 degrees, the hole showing the most pixel "matches" is most likely the topmost object.

This concludes the basic overview of the six stages in the overlapping objects recognition system. More detailed algorithms for the system actually implemented are provided in the next section. Those readers not interested in specific details may skip this section and jump to the section on results and conclusions with no loss in continuity.

3. Detailed Algorithms

This section of the report provides the detailed algorithms for PHASE 1 and all six stages in PHASE 2 of the overlapping objects recognition system. This section can be omitted with no loss in continuity.

3.1. PHASE 1 - Training

The Training phase of this project focuses primarily on the large hole in the center of the object. Using this hole, a radial template is calculated and then stored in a file for future use. Each template value contains the distance from the center of the hole to the outer edge of the object for each one degree increment. The algorithm for the training is as follows:

STEP 1

Process the image of the single object using the algorithms for edge detection, noise elimination, border creation and labeling.

STEP 2

Process the image using the algorithm for feature recognition in order to find the hole in the center of the object.

STEP 3

Starting at the center of the hole, measure the distance to the outer edge of the object at one degree increments.

STEP 4

Store all 360 of these measured lengths (radial template) in a disk file for future use.

STEP 5

Repeat steps 1-4 for all the stable states of the object.

3.2. PHASE 2 - Object Recognition

3.2.1. Image Acquisition Stage

The first stage in PHASE 2 of the system is to acquire an image of the overlapping parts. This is done by taking a picture of the objects in their overlapping state using a Hamamatsu T.V. Camera and running it through a high-speed A/D converter as described in Appendix C. For development purposes, the resolution of each picture was standardized to 256×256×8 bits.

3.2.2. Edge Detection Stage

The second stage in the recognition of overlapping objects is the edge detection stage.

3.2.2.1. Gradient Calculation

The first step in the edge detection stage is to determine the location of all possible edges using a gradient magnitude calculation. The procedure is as follows:

```
(1) Initialize: i=j=1

(2) Read a 3x3 window
    (Read a 5x5 window)

(3) MAGX = pixel[i-1][j] - pixel[i+1][j]
    MAGY = pixel[i][j-1] - pixel[i][j+1]
    (MAGX = pixel[i-2][j] - pixel[i+2][j] )
    (MAGY = pixel[i][j-2] - pixel[i][j+2] )

(4) Total = Sqrt(MAGX*MAGX + MAGY*MAGY)

(5) Normalize the result to the range (0-255)
    Total = Total/1.414

(6) Flag central pixel if it is a potential edge, otherwise, turn it off
    If (Total ≥ Threshold) OR (MAGX ≥ X-thresh) OR (MAGY ≥ Y-thresh)
        Then pixel[i][j] = 255
        Else pixel[i][j] = 0

(7) Repeat steps (2)-(6) for all 256x256 pixels.
```

3.2.2.2. Noise Elimination

The second step in the edge detection algorithm is to eliminate any extraneous noise picked up in the gradient calculation step. The procedure is as follows:

```
(1) Initialize: i=j=1

(2) Look for central pixel:
    If pixel[i][j]=0 then goto step (6)

(3) Read a 3x3 window

(4) Count the number of logical 1 pixels in the window:
    Count= pixel[i-1][j-1] + pixel[i-1][j] +
        pixel[i-1][j+1] + pixel[i][j-1] +
        pixel[i][j+1] + pixel[i+1][j-1] +
        pixel[i+1][j] + pixel[i+1][j+1]
```

```
(5) Eliminate pixel if it is a noise pixel:
    If count ≤ Noise-threshold then pixel[i][j] = 0

(6) Repeat steps (2) - (5) for all 256x256 pixels
```

3.2.2.3. Border Creation

The third step in the edge detection algorithm is to create a bounding rectangle, or window, around the overlapping objects. The procedure is as follows:

```
(1) Initialize:
    i=j=1
    XMIN=YMIN=256
    XMAX=YMAX=1

(2) Assign minima and maxima:
    If pixel[i][j] = 255
  then if  i < XMIN then XMIN = i
        if  i > XMAX then XMAX = i
        if  j < YMIN then YMIN = j
        if  j > YMAX then YMAX = j

(3) Repeat step (2) for all 256x256 pixels

(4) Draw bounding rectangle using XMIN, XMAX, YMIN, YMAX
```

3.2.3. Labeling Stage

The labeling technique, which segments and labels the binary image, makes use of two simple data structures, and also a row transition data structure. All three data structures are implemented using a 200 element, one dimensional array. This was considered to be more than adequate for processing a 256x256 pixel image. The arrays are called startc (starting column), component (component number), and trans (transitions).

As mentioned before, rows are divided into segments before being processed. Each segment in the row is assigned some component number, representing which part of the object it is. In this way, the startc and component data structures are related to each other. For example, assume that these data structures are in the following states:

Startc	0	10	26	255
Component	0	1	2	dummy

This would mean that in the old row we have three segments. The first runs from column 0 up to but not including column 10. This segment has been assigned the component number "0". The second segment runs from column 10 to column 26, and is assigned the component number "1". The third segment runs from column 26 to column 255, and is assigned the component number "2".

To explain how the labeling technique is implemented, we should first define some terminology. Then, we will explain the action taken in each of the four cases which occur in the algorithm. Finally, we will examine a sample image and run through the algorithm, showing the data structures and the labeled image at various stages throughout the procedure.

3.2.3.1. Labeling Terminology

Current Pointer - this is a pointer to an element in the startc and component data structures that points to the current segment and its corresponding component number.

Old Row - this is the row which is defined by the startc and component data structures.

New Row - this is the row which is currently being processed in the binary image.

Current Segment - this is the segment which is pointed to by the current pointer to the startc data structure. It is the segment which the new segment will be compared to in order to recognize one of the four cases.

New Segment - this is the segment being looked at in the new row. It is compared to the current segment in the startc data structure.

Previous Segment - this is the segment just before the current segment in the startc data structure.

Subsequent Segment - this is the segment just after the current segment in the startc data structure.

Transition - a 0 to 1 or a 1 to 0 transition in the binary image.

Labeled Array - this is the final 256x256 array which contains the labeled pixels.

3.2.3.2. The four labeling cases

In analyzing each row, we run across four separate cases. They are as follows:

Case 1: New segment lies to the right of the current segment

If Case 1 is recognized, the startc and component data structures are too big. Therefore, a segment must be deleted from them and neighboring segments may have to be merged. In other words, two parts of the object which were thought to be separate parts are actually connected and should be labeled with the same component number. The action taken in Case 1 is:

STEP 1

Remove the current segment from the startc data structure.

STEP 2

If the component numbers of the previous segment and the subsequent segment are the same, GOTO STEP 4.

STEP 3

If the component numbers of the previous segment and the subsequent segment are different, change all occurrences of one to the other in both the component data structure and the labeled array. Keep track of what the background is assigned if it is changed by this action.

STEP 4

Remove the component numbers which corresponded to the current segment and the subsequent segment from the component data structure.

STEP 5

Continue processing.

Case 2: New segment lies to the left of the current segment

If Case 2 is recognized, the startc and component data structures must be increased in size, and the new segment added to them. In other words, a new part of the object has been found and needs to be assigned a new component number. The action taken in Case 2 is:

STEP 1

Insert the starting and ending column numbers of the new segment into the startc data structure right before the current segment.

STEP 2

Choose an unused component number for the new segment and insert it into the component data structure.

STEP 3

Choose the component number of the previous segment to be the component number of the subsequent segment, and insert it into the component data structure in the position just after that pointed to by the current pointer.

STEP 4

Continue processing as if a Case 3 overlap was recognized.

Case 3: New segment and the current segment overlap.

If Case 3 is recognized, the first thing that must be checked is if the new segment is the final segment in the new row, and if so, whether it is being assigned a component number other than the background number. If so, then the startc and component data structures are not entirely correct, and Case 4 must be processed before Case 3. If not, the following action is taken:

STEP 1

The new segment is assigned the same component number as the current segment in the startc data structure.

STEP 2

The starting column of the new segment replaces the starting column of the current segment in the startc data structure.

STEP 3

The pixels in the labeled array corresponding to the new segment in the new row are set to the component number of the new segment.

Case 4: The data structures are not entirely correct

If Case 4 is recognized, this indicates that the startc and component data structures contain excess information which is not needed, and must be changed accordingly. This case results when the final segment in the new row is going to be assigned a component number other than the background number. The action taken is the following:

STEP 1

Record the value of the component number which corresponds to the current segment.

STEP 2

Change all occurrences of this component number in the component data structure and in the labeled array to the background component number.

STEP 3

Remove all elements from the startc and component data structures, starting with the element after the element which is pointed to by the current pointer, up to but not including the last element in the data structure.

STEP 4

Continue processing with Case 3.

3.2.3.3. A labeling example

To clarify the labeling procedure, let us look at an example, and run through the procedure. Suppose we began with a 8×10 binary image as shown in Figure 12 below.

						Columns					
		0	1	2	3	4	5	6	7	8	9
	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	1	1	0	0	0	0
	2	0	0	0	1	1	1	1	0	0	0
Rows	3	0	0	0	1	0	0	1	0	0	0
	4	0	0	0	1	1	1	1	1	1	0
	5	0	0	0	0	1	1	1	0	1	0
	6	0	0	0	0	1	1	1	0	0	0
	7	0	0	0	0	0	0	0	0	0	0

Figure 12 An 8×10 Binary Image Example

When analyzing this binary image, we first find all the transitions in each row and record the column numbers where they occur.

Row 0	-	no transitions
Row 1	-	Columns 4,6
Row 2	-	Columns 3,7
Row 3	-	Columns 3,4,6,7
Row 4	-	Columns 3,9
Row 5	-	Columns 4,7,8,9
Row 6	-	Columns 4,7
Row 7	-	no transitions

We start by initializing the startc data structure to contain a segment with the starting column number as 0 and the ending column number which is one more than the highest column number in the binary image. We give this segment the component number "0" (background), and set the current pointer to point to it.

```
Startc          0     10

Component       0     dummy
                ↑
             Current
             Pointer
```

Now we look at row 0. It has no transitions, which means it consists of one segment of all 0's, which runs from column 0 up to (but not including) column 10. This is the new segment. The current segment also runs from column 0 to column 10, so this results in a Case 3 overlap. Case 3 results in the data structures remaining unchanged. The labeled array and the data structures now look as follows:

```
                Row 0     0000000000

Startc          0     10
Component       0     dummy
                      ↑
```

Since there are no more segments in the new row, we reset the current pointer to point to the first element in the startc data structure, and process the next row. Row 1 has transitions at columns 4 and 6. This results in the new row (Row 1) having 3 segments (0-4), (4-6), and (6-10). Comparing the new segment (0-4) with the current segment (0-10), we get a Case 3 overlap. The next segment in the new row runs from columns 4 to 6, but this lies to the left of the current segment, which starts in column 10. Therefore, Case 2 must be processed, and the new segment added to the data structure. An unused component number is chosen for the new segment, and the labeled array and data structures look as follows:

```
                      Row 0     0000000000
                      Row 1     000011

        Startc        0    4    6    10
        Component     0    1    0    dummy
                                ↑
```

The last segment in the new row runs from column 6 to column 10, and results in another Case 3 overlap. We now have:

```
                      Row 0     0000000000
                      Row 1     0000110000

        Startc        0    4    6    10
        Component     0    1    0    dummy
                                     ↑
```

We now go on to row 2. This has transitions at columns 3 and 7. This gives us 3 segments to look at (0-3), (3-7), and (7-10), all of which result in Case 3 overlaps. This gives us the following situation:

```
                 Row 0    0000000000
                 Row 1    0000110000
                 Row 2    0001111000

      Startc     0    3    7    10
      Component  0    1    0    dummy
                                 ↑
```

Row 3 has transitions at columns 3, 4, 6 and 7, giving 5 segments (0-3), (3-4), (4-6), (6-7) and (7-10). The first two segments result in Case 3 overlaps, but the third segment (4-6) results in Case 2 when compared to the current segment (7-10). This causes the segment (4-6) to be added to the data structure and gives the following:

```
                 Row 0    0000000000
                 Row 1    0000110000
                 Row 2    0001111000
                 Row 3    000122

      Startc     0    3    4    6    7    10
      Component  0    1    2    1    0    dummy
                                ↑
```

We now continue processing the segments (6-7) and (7-10), which results in two more Case 3 overlaps. We now have:

```
                Row 0      0000000000
                Row 1      0000110000
                Row 2      0001111000
                Row 3      0001221000

Startc          0     3     4     6     7     10
Component       0     1     2     1     0     dummy
                                               ↑
```

Row 4 contains the 3 segments (0-3), (3-9) and (9-10). For the first two segments we get Case 3 overlaps, but when comparing the new segment (9-10) with the current segment (4-6), we get a Case 1. This means we need to merge 2 elements in the startc data structure and remove the current segment from it. When we first come across this Case 1, we have:

```
                Row 0      0000000000
                Row 1      0000110000
                Row 2      0001111000
                Row 3      0001221000
                Row 4      000111111

Startc          0     3     4     6     7     10
Component       0     1     2     1     0     dummy
                            ↑
```

After processing Case 1 and merging, we have:

```
                 Row 0     0000000000
                 Row 1     0000110000
                 Row 2     0001111000
                 Row 3     0001221000
                 Row 4     000111111

    Startc       0    3    7     10
    Component    0    1    0     dummy
                           ↑
```

Since the component numbers of the segments before and after the current segment were both "1", we did not have to go to the labeled array and component data structure to merge one component to the other. Finally, we get another Case 3 overlap to finish row 4, and this gives us:

```
                 Row 0     0000000000
                 Row 1     0000110000
                 Row 2     0001111000
                 Row 3     0001221000
                 Row 4     0001111110

    Startc       0    3    9     10
    Component    0    1    0     dummy
                                 ↑
```

Row 5 has transitions at columns 4, 7, 8 and 9 giving the five segments (0-4), (4-7), (7-8), (8-9) and (9-10). The first two result in Case 3 overlaps. The third results in a Case 2, causing the new segment (7-8) to be added to the data structures. The last two segments also result in Case 3 overlaps, and that finishes row 5. We now have:

```
                    Row 0     0000000000
                    Row 1     0000110000
                    Row 2     0001111000
                    Row 3     0001221000
                    Row 4     0001111110
                    Row 5     0000111310

     Startc         0    4    7    8    9    10
     Component      0    1    3    1    0    dummy
                                              ↑
```

Row 6 has 2 transitions at columns 4 and 7. This gives three segments to look at (0-4), (4-7) and (7-10). All three result in Case 3 overlaps. However, it is important to notice that when we compare the new segment (7-10) with the current segment (7-8), we will be setting the final segment in the new row to a component number "3" other than the background "0". This results in Case 4, and the startc and component data structures contain erroneous information. We must also change all occurrences of the component number "3" in the labeled array and the component data structure to "0". Thus, processing Case 4, we get:

```
                Row 0     0000000000
                Row 1     0000110000
                Row 2     0001111000
                Row 3     0001221000
                Row 4     0001111110
                Row 5     0000111010
                Row 6     0000111000

Startc          0    4    7    10
Component       0    1    0    dummy
                                 ↑
```

Row 7 has no transitions, and thus one segment (0-10). This results in a Case 3 overlap and is assigned the component number "0". Since we found a row with no transitions, we also re-initialize the data structures, and now have:

```
                Row 0     0000000000
                Row 1     0000110000
                Row 2     0001111000
                Row 3     0001221000
                Row 4     0001111110
                Row 5     0000111010
                Row 6     0000111000
                Row 7     0000000000

Startc          0    10
Component       0    dummy
                       ↑
```

Thus, the object has been labeled into three different parts (components). Component "0" is considered to be the background, component "1" is the actual object, and component number "2" could be a hole in the object.

3.2.4. Feature Recognition Stage

At this point in the processing, the feature recognition stage will search for all possible holes in the objects. This is done using three tests on the possible holes. They are 1) the area test, 2) the ratio test, and 3) the continuity test. If a possible hole passes all three tests, it is considered to be a hole. The algorithm used is as follows:

STEP 1

Using information from the labeling stage, find all component numbers such that between 90 and 160 pixels in the labeled array were labeled with that component number (area test).

STEP 2

For each component number which meets the area test, count how many columns in the labeled array have at least one occurrence of that component number.

STEP 3

For each component number which meets the area test, count how many rows in the labeled array have at least one occurrence of that component number.

STEP 4

Divide the number of rows (STEP 2) by the number of columns (STEP 3). If this ratio falls within a given range (usually 0.8-1.2), then this passes the (ratio test).

STEP 5

For all possible holes which pass the first two tests, check along the principle horizontal axis for transitions from the component number being tested to another component number or vice versa. If there are two transitions, the continuity test is satisfied.

STEP 6

Store the component number of each possible hole that passes all three tests.

These are now considered to be holes.

3.2.5. Template Matching Stage

In order to reconstruct the image of the overlapping objects, we need to process the template matching algorithm. This enables us to determine the orientation of each object by comparing information taken from the labeled picture with information from a stored radial template. The algorithm is as follows:

STEP 1

Find the centers of each of the holes which passed the three tests in the feature recognition stage (cx[i],cy[i]).

STEP 2

Input the increment in degrees that is to be used in the template matching. Call this "increment".

STEP 3

Measure the distance from the center of the hole to the background, starting at an angle of 0 degrees, and measuring every "increment" degrees (Length[i], i=0 to 359 step "increment").

STEP 4

Read the radial template from the file (Known[i], i=0 to 359)

STEP 5

Match Length[i] to Known[i], and let the total deviation be:

$$\textit{total deviation} = \sum_{i=0}^{359} \left(\textit{Length}[i] - \textit{Known}[i] \right)^2$$

However, do not add the i^{th} deviation if:

$$\textit{Length}[i] > (1.5) \times \textit{Known}[i]$$

or

$$Length[i] < (0.5) \times Known[i]$$

STEP 6

Rotate the Known template "increment" degrees, and repeat STEP 5.

STEP 7

Repeat STEP 5 and STEP 6 until the Known template has been rotated 360 degrees. Always keep a record of the lowest value for the deviation and what the orientation of the Known template was when this lowest value for deviation was calculated.

STEP 8

Repeat STEPS 4-7 for all of the templates which were trained during the Training phase.

STEP 9

The lowest deviation will now indicate what the orientation of the object is and which stable state it is in.

STEP 10

Repeat STEPS 3-9 for each hole found in the feature recognition stage.

3.2.6. Objects Recognition and Reconstruction Stage

After the image has been labeled and processed by the feature recognition and template matching stages, it can then be processed by the objects recognition and reconstruction stage. For each object in which the central hole was found, we can reconstruct the object using the trained template and the position and orientation found in the template matching stage. Then, by comparing or correlating this reconstructed image with the edge detected image, we can determine which object is most likely to the topmost object using the following algorithm:

STEP 1

Reconstruct the object by using the trained template and the position and orientation found in the template matching stage. Set the correlation counter for the object to 0.

STEP 2

For each "on" (logical 1) pixel that is part of the reconstructed image, if that pixel (or a pixel within a very small neighborhood of that pixel) is "on" in the edge detected image, then increment the correlation counter for the object by 1.

STEP 3

Repeat STEPS 1-2 for all objects of which the central hole was found in the feature recognition stage.

STEP 4

Compare the highest correlation count obtained with a correlation threshold. If the count is above the threshold, then the object with this correlation is the topmost object. If not, then the algorithm has failed to produce a topmost object.

4. Results and Conclusions

This section discusses the results obtained in PHASE 1 and all six stages of PHASE 2 in the overlapping objects recognition system. It also discusses any advantages, limitations, and possible improvements to the various algorithms.

4.1. PHASE 1 - Training

The first phase in the system is the Training phase. It involves calculating a radial template for each object in all its non-overlapping stable states.

The results of this phase amounted to simple arrays containing distance values from the center hole to the outside edge for all 360 degrees. One advantage of this is that it allows the system to become more automated and require less human inspection. That is, it helps to make the system more object independent so that it can be used for a variety of objects and not just for those shown in appendix A. Another advantage is that it only has to be run once for each object in all its stable states. Once the distances are calculated they never have to be calculated again.

One way to improve the Training phase is to make it even more object independent. Instead of locating only the circular hole in the center of the object, the Training phase can be upgraded so that it is smart enough to locate other local features. For example, have it locate holes of different sizes and shapes, and perhaps even edges that have sharp corners or smooth curves.

4.2. PHASE 2 - Object Recognition

The second phase in the system is the actual recognition and reconstruction of overlapping objects in their overlapping state.

4.2.1. Image Acquisition Stage

The first stage in the Object Recognition phase is the actual acquisition of the overlapping objects image.

This phase is critical to the success of all future stages in the system. Appendix C shows the results of various images after they were acquired and digitized. Picture 1 shows an image where only one of the two holes are non-overlapped. Picture 2 and 4 show images where both holes are non- overlapped. Picture 3, on the other hand, shows images where both holes are overlapped and hidden.

4.2.2. Edge Detection Stage

The second stage in the recognition and reconstruction of overlapping objects is to determine the location of all possible edge pixels in the overlapping objects image.

Most images processed with this algorithm resulted in very good boundary/edge maps (see Appendix E). However, lighting was observed to be a major factor in the success. After considerable experimentation, the base light was found to be an absolute necessity. It blocked out reflections from the overhead light and other intruding light sources. When both side lights were left off or turned on, the algorithm produced excellent outer edges, but only a trace of the overlapping edge. In any case, the best results were obtained when only one of the two side lights were turned on. This seemed to produce a shadow on the overlapping edge that could be detected by the gradient technique. The question of which side light to turn on seemed unimportant to the algorithm. In fact, both produced nearly identical results.

In any case, the lighting procedure is critical to the algorithm. Many questions still need to be answered. For example, should a higher light intensity be used? Should it be projected at a different angle, perhaps from almost directly overhead? Should fluorescent lights be used? In any event, the idea is to create a more intense shadow on the overlapping edge in order to increase its detectability. That is, to optimize the edge detection algorithm.

While the gradient calculation step of this algorithm flagged many edge pixels, it also flagged many non-edge or "false pixels" as edges. The noise elimination step of this algorithm attempts to eliminate as many of these "extraneous" pixels as possible. The results were fairly successful. Refer to Appendix D for images before the noise elimination and Appendix E for images after the noise elimination.

Overall, the edge detection algorithm had a fast processing time, about 45-60 seconds under "optimal conditions", that is, when a small number of users were on the system. The 45-60 seconds included reading the digitized pixel data from the disk file into a 256×256 array, calculating the magnitude of the gradient at each pixel and determining which pixels were edge pixels and which were extraneous noise, and displaying the edge detected map on the Ramtek monitor. The gradient calculation accounted for about 80% of the total time, while the noise elimination, border creation and image display combined for the remaining 20%.

4.2.3. Labeling Stage

The third stage in the recognition of overlapping objects is to separate the image into its individual components using the labeling technique.

The components of the image that could be separated by this algorithm for the two objects used in the project are the:

(1) background
(2) outside edge
(3) overlapping edge
(4) inside area of the objects
(5) outside edge of any holes
(6) inside area of any holes

Appendix F shows the individual components that were separated for one particular overlapping objects image.

The background component is probably the easiest to distinguish; it has the largest area and involves the most pixels. Any transition in this component marks an edge pixel. Because of this feature, the background component is the ideal candidate for the template matching stage.

Ideally, we would like to get both the outside edge and the overlapping edge all labeled with the same component number, and the inside area of the two objects labeled with different component numbers. However, since the overlapping edge was never completely solid and connected in any of the images after the edge detection stage, both objects became labeled as one object (with one component number). Thus, one improvement in the system would be to have an additional stage just before the labeling that somehow connects the overlapping edge with itself and also with the outside edge. This will force each object to be uniquely labeled (with different component numbers) and be clearly distinguishable.

In any case, the inside area of the holes and the outside border of the holes came out very well. In fact, in every image tested, the area of the internal hole ended up to be approximately the same value.

The main advantage of the labeling technique is its fast processing time. During "optimal conditions" (a low number of system users) it takes about 15-20 seconds to process a 256x256 pixel binary image like those shown in Appendix E. The algorithm is faster than the edge detection stage for two reasons. First, instead of having to process one pixel at a time, a whole group of pixels or segments can be processed together. Second, whereas the edge detection stage had to process all 256x256 pixels, the labeling technique only needs to consider those pixels that reside within the window created by the edge detection stage.

4.2.4. Feature Recognition Stage

The fourth stage in the recognition of overlapping objects is to scan through the labeled images looking for certain characteristics or identifying features.

The feature that this algorithm concentrated on was the relatively large circular hole in the center of the object. If the hole was not overlapped by another object it could almost always be found. For example, of the images shown in Appendix B, both holes could be located in pictures 2 and 4, one hole in picture 1, and neither hole in picture 3. Thus, at present, the recognition is completely dependent on the visibility of the holes. An obvious improvement would be to develop a procedure to be able to recognize other features such as sharp corners and smooth edges. This will considerably increase the flexibility of the system.

One limitation of this algorithm, however, is that if the objects are positioned so that they form a pseudohole, or a hole that is not really there, the algorithm will still think it is a hole (see Figure 13). This error could, however, be recognized in the next stage when radial templates are compared and no suitable match is found.

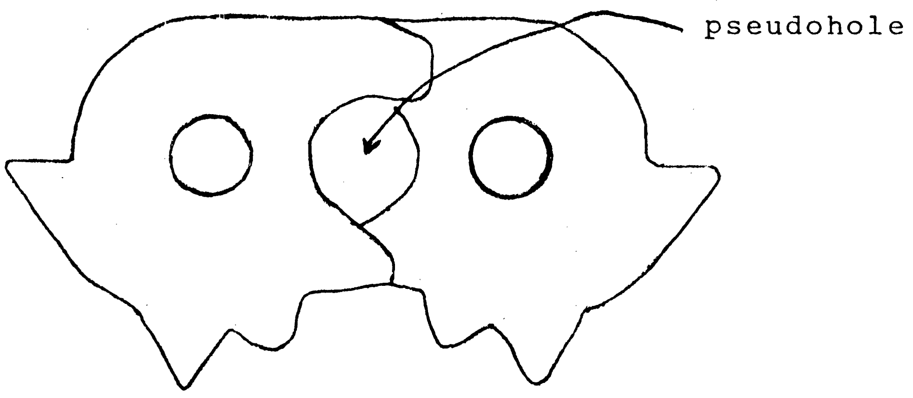


Figure 13 Formation of a Pseudo-hole

4.2.5. Template Matching Stage

The fifth stage in the recognition of overlapping objects is to match the radial template(s) calculated in the Training phase with the radial template(s) calculated for every hole to determine the orientation of the objects.

The results of this stage were very good. For every hole that was located in the previous stage, the algorithm could determine the "best fit" orientation. For the images shown in Appendix E, the "best fit" orientation was in fact the correct orientation.

Note, however, that the entire template matching algorithm is completely dependent on the visibility of the holes. Without them, there is simply no starting place for the algorithm to run. The algorithm could be strengthened if more features were detectable in the previous stage. This would alleviate the problem of having to rely solely on the hole. In addition, if other features such as sharp corners could be found, they could also be located in the template to yield the orientation.

In the case where two holes plus a pseudohole are found, the template matching stage can easily eliminate the pseudohole. The pseudohole is simply the hole whose "best fit" orientation is weakest compared to the other two.

The time required for the template matching stage varied depending on the degree increment by which the matching was done. The smaller the increment, the more accurate the "best fit" orientation turned out to be. However, the algorithm required a tradeoff between accuracy and time. An image where two holes were found required 15-25 seconds to match with a one degree increment, and only about 5-10 seconds with a three degree increment. In almost all cases, increments of up to five degrees provided enough accuracy to find the "best fit" orientation.

4.2.6. Objects Recognition and Reconstruction Stage

The sixth and last stage in the recognition of overlapping objects is to make the final recognition and reconstruct any hidden or overlapped edges.

For two overlapping objects, this stage in the system was very successful provided both holes were accurately located in the previous stages. If they were, the system could determine the position of the objects (x & y coordinates of the center of the holes), and the orientation of the objects (number of degrees rotated relative to the trained object), and identify which object was on top. If, however, only one hole was located, only the position and orientation of that particular object could be determined. No information about the other object could be determined. If no holes were found, then nothing could be determined about either object.

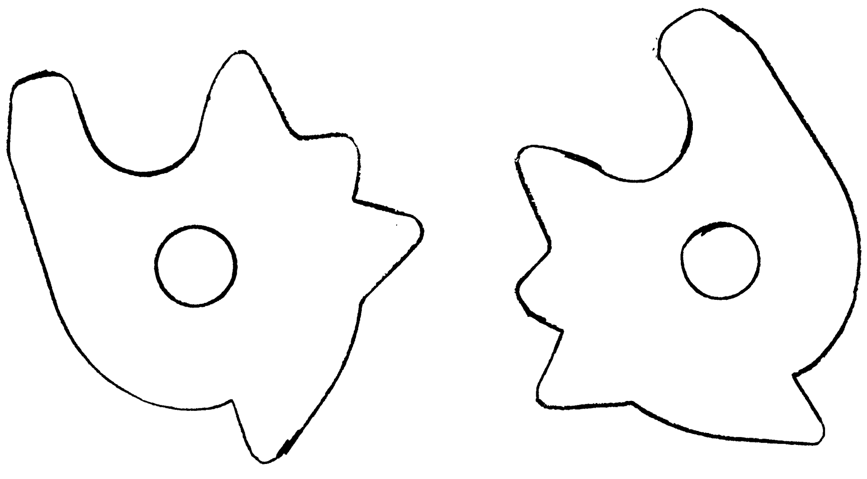
One possible solution in the case where one hole is found is to have the robot pick up that object, and then run the algorithm over again starting from the image acquisition stage. By removing the first object, the hole for the second object is exposed and, thus, can be detected on the second try. The only problem is that we cannot tell if the first object was really on top or not.

4.3. Conclusion

The computer vision system presented in this report can determine the position and orientation of two overlapping objects. First, the Training phase teaches the computer to recognize the objects in all of their non-overlapping stable states. Second, the image acquisition stage converts the analog image obtained from a single T.V. camera into an 8-bit grey-scale image of the the overlapping parts. Third, the edge detection stage converts the grey-scale image into a binary image while locating as many edge pixels as possible. Fourth, the labeling technique segments the binary image by labeling its various internal and external components. Fifth, the feature recognition stage searches the labeled image for local features such as holes, and determines the position of their corresponding objects. Sixth, the template matching stage computes a radial template, compares it with templates generated in the Training phase, and determines the orientation of the objects. And last, the objects recognition and reconstruction stage reconstructs the objects and determines which one is on top. The only requirements are that the edge detection stage generate a "good" internal edge, the objects have some clearly distinguishable features such as holes, and that at least one of these features be non- overlapped. For the objects used in this project, the vision system was unable to recognize only those objects whose internal hole was "hidden"; it could still recognize all the other objects. However, to accurately determine which object is on top requires all of the holes to be completely visible.

6. Appendix A: Objects Used

The objects used in this research project were identical, unsymmetrical, metallic door lock parts shown below in actual size.



Objects Used in the Current Research.

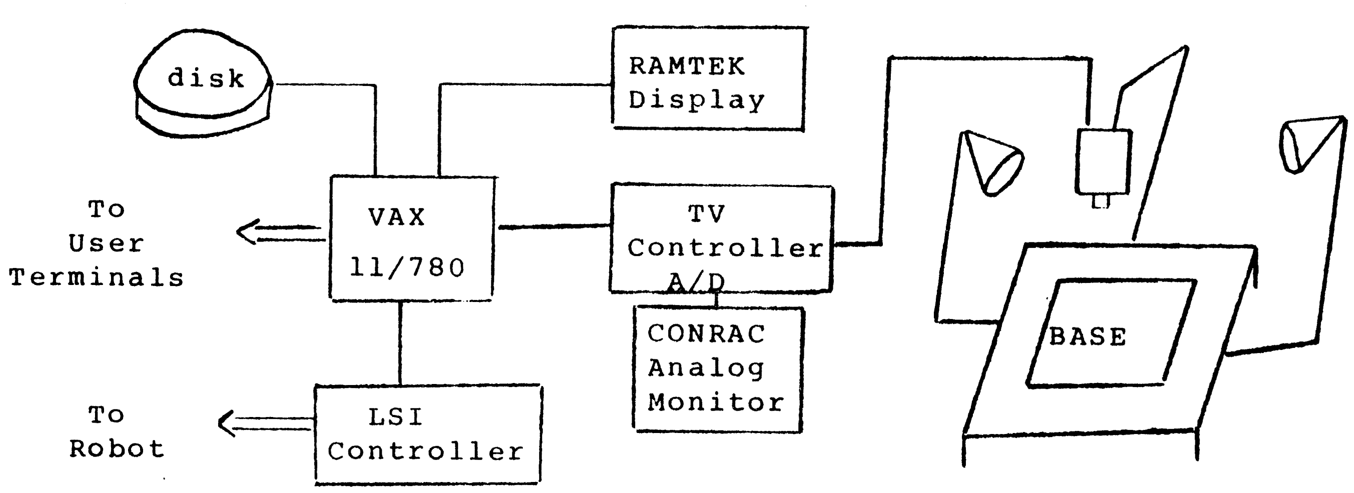
These parts have many of the characteristic features found in other industrial parts and are well suited for the bin-picking application. They have 4 sharp corners, 2 rounded corners, 1 straight edge, many curved edges, 1 circular hole, and 1 small circular indentation.

6. Appendix B: Equipment Used

The algorithms were developed and tested on a VAX 11/780 four mega-byte Computer System and a Ramtek Graphic Display System, using the C-Programming language, the Picture Processing System (PPS), and the Unix operating system.

A Hamamatsu T.V. camera, high-speed Analog/Digital converter, and a Conrac analog monitor were used to take pictures of small industrial components with various degrees of overlap. For simplicity and ease of processing, each picture was converted into a 256x256x8-bit grey-scale image using a single scan line.

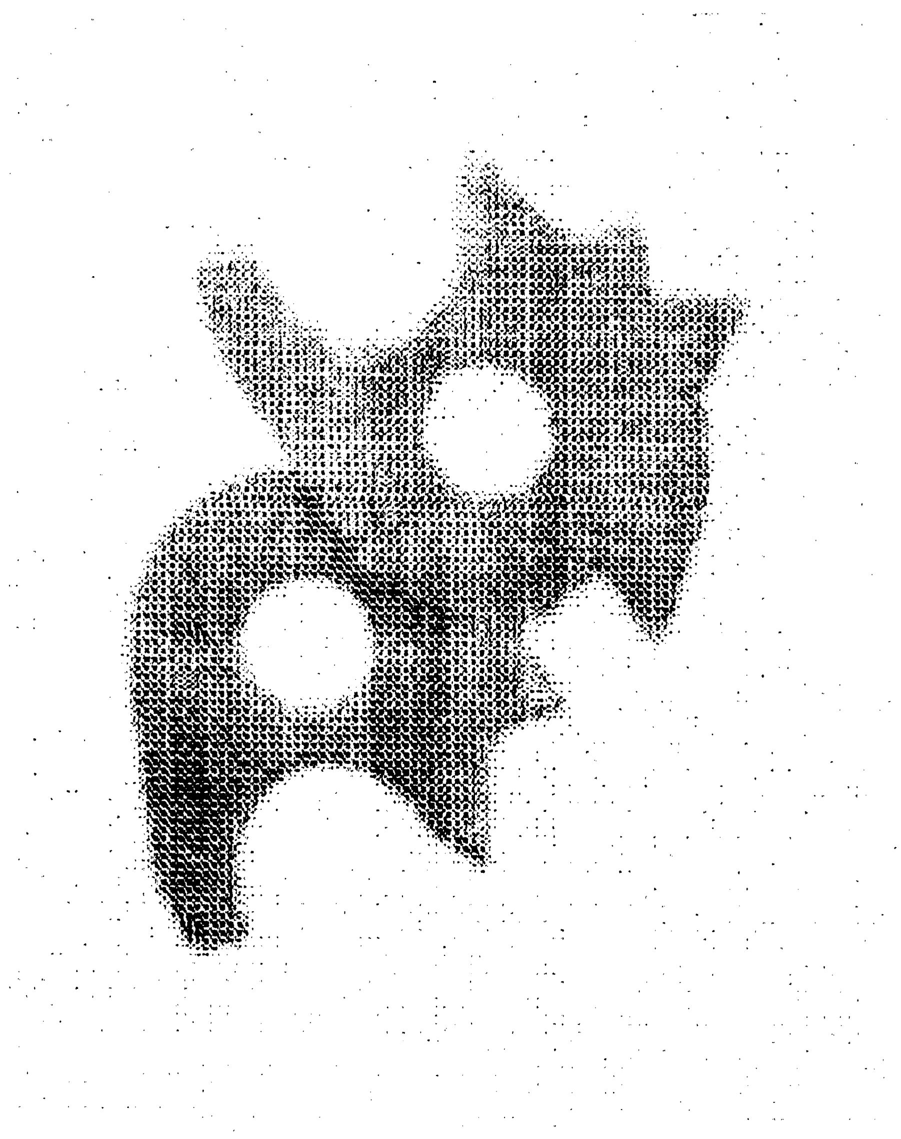
The T.V. camera was mounted vertically just above the lighted work-base with additional lights on each side. The complete system is shown below.



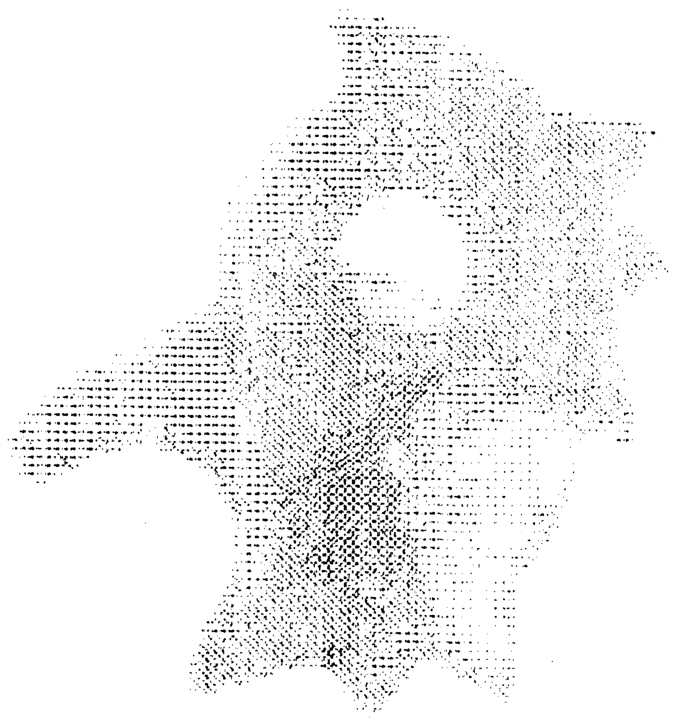
7. Appendix C: Digitized Images

This appendix shows the digitized images of two overlapping objects before any processing was performed.

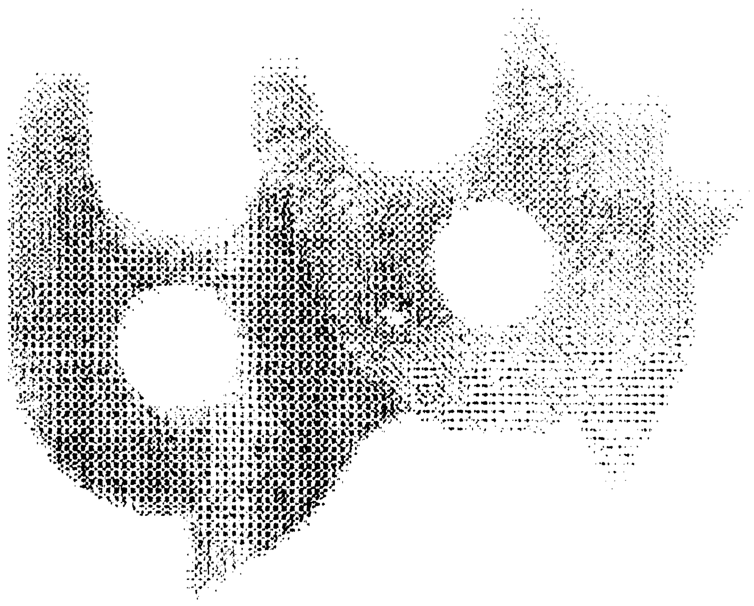
Picture 1



Picture 2



Picture 3

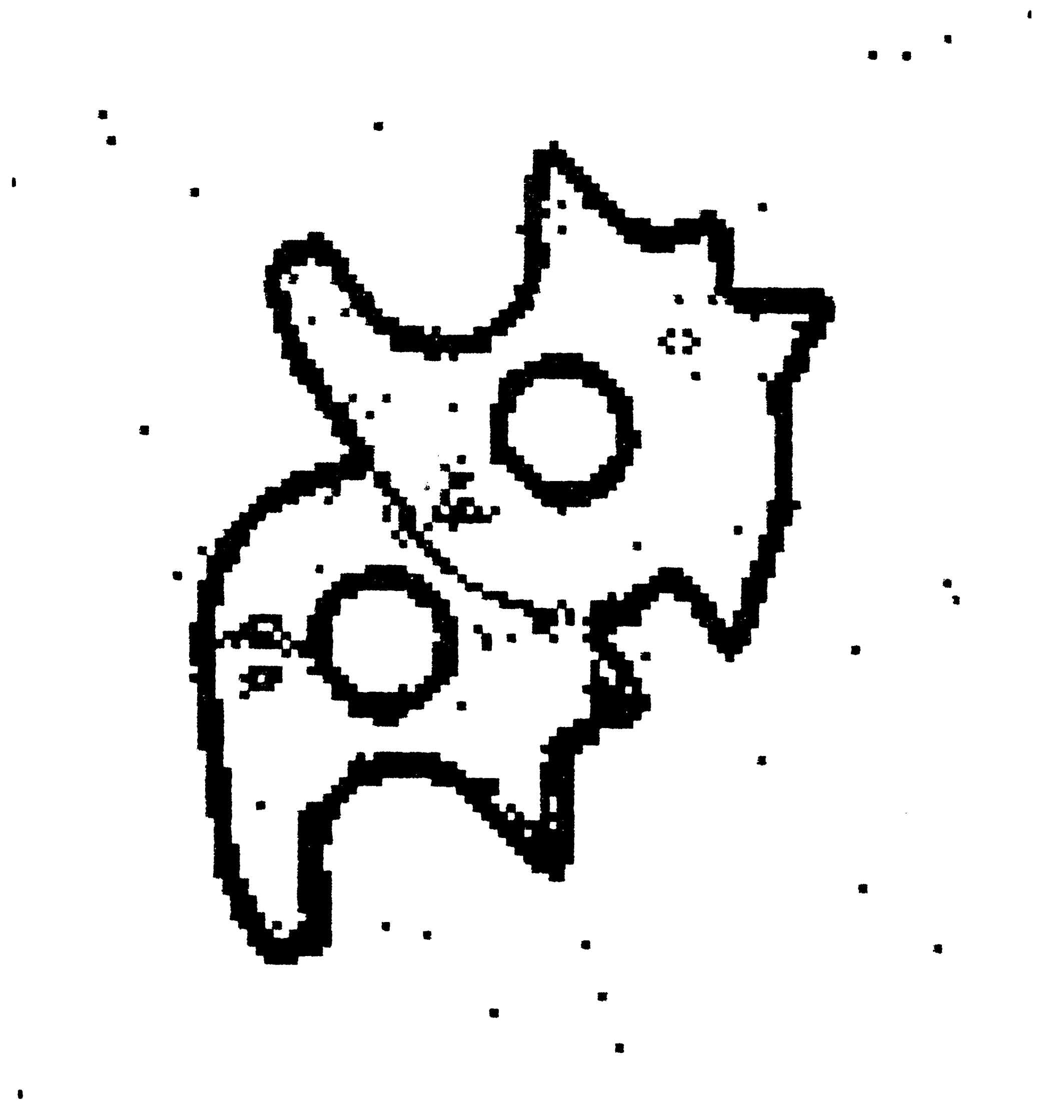


Picture 4

8. Appendix D: Edge Detected Images

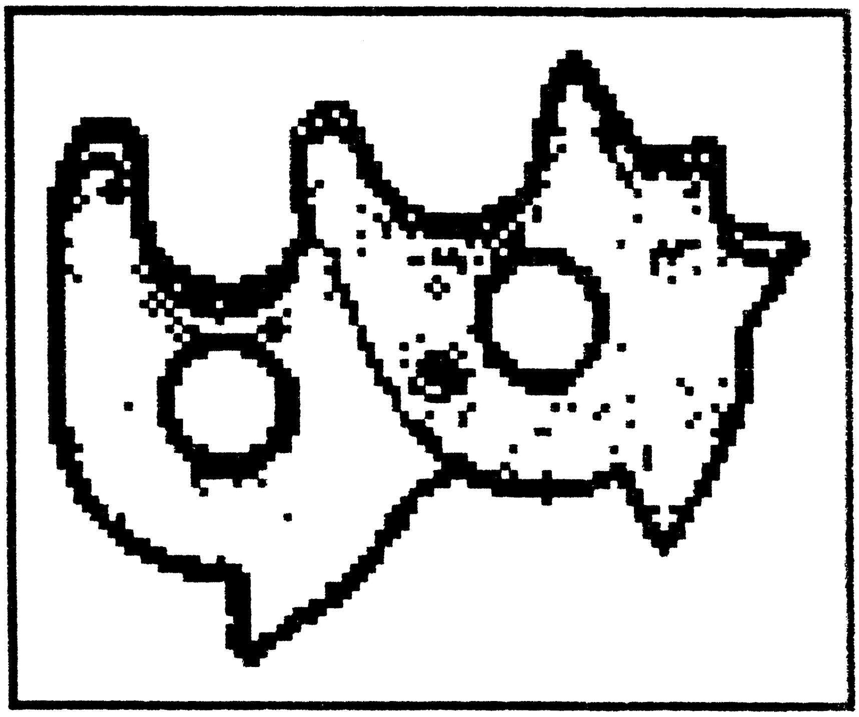
This appendix shows the processed images after the gradient calculation step of the edge detection algorithm but before the noise elimination step.

Picture 1



Picture 2

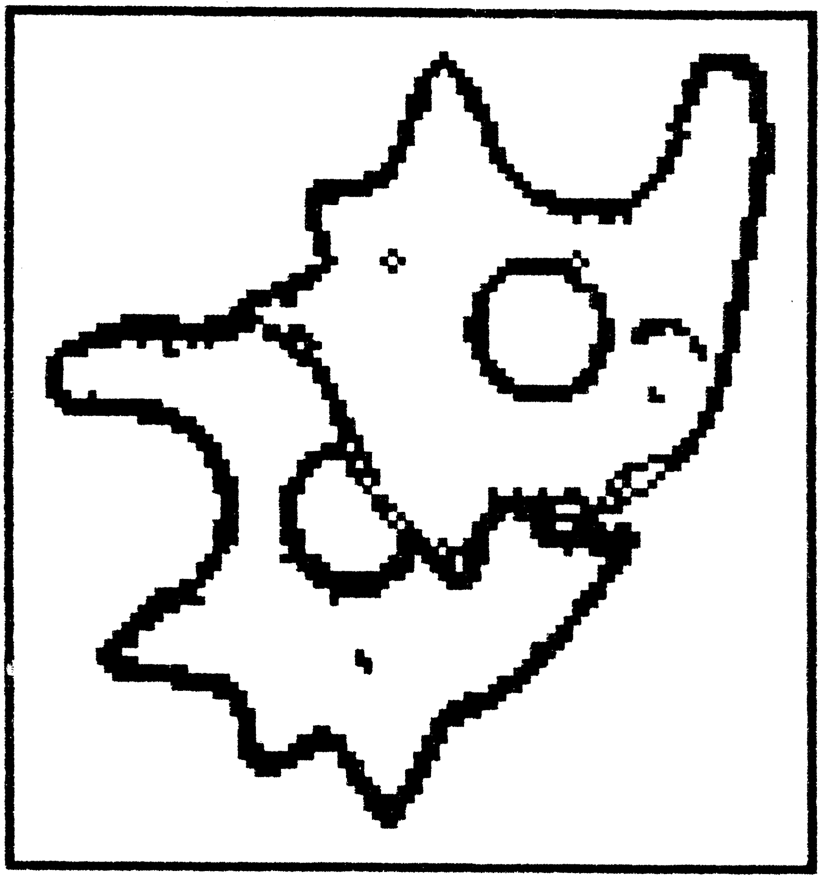
Picture 3



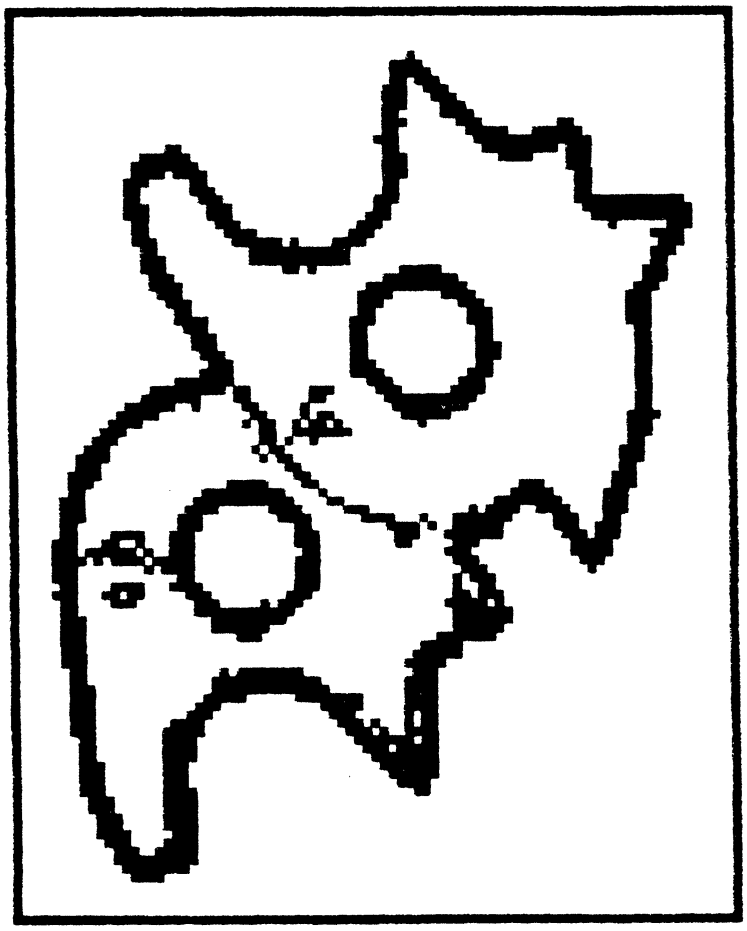
__Picture 4__

9. Appendix E: Noise Eliminated Images

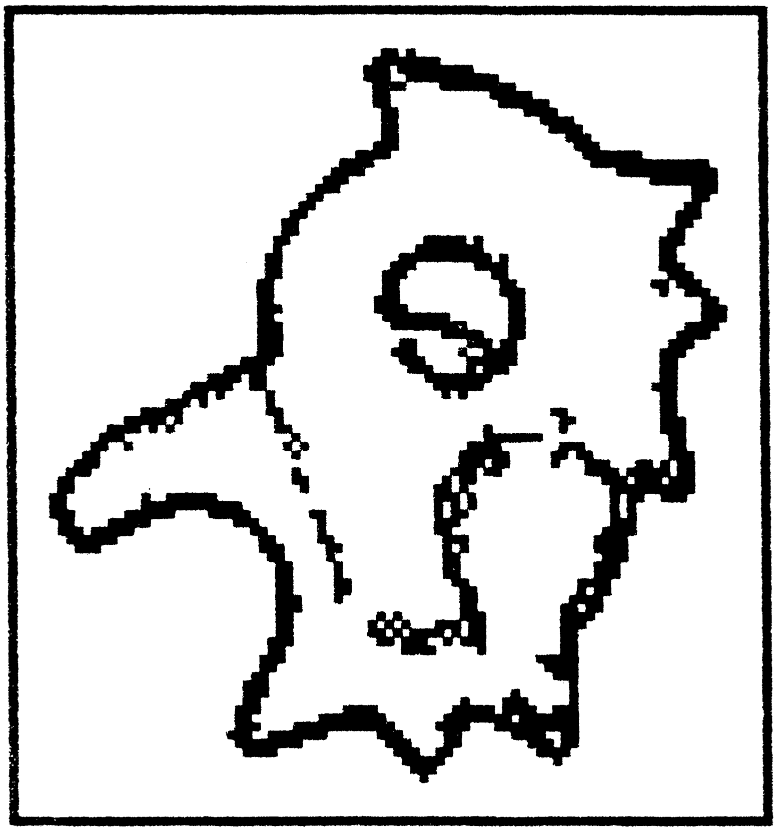
This appendix shows the processed images after the gradient calculation step (step 1), the noise elimination step (step 2), and the border creation step (step 3) of the edge-detection stage.



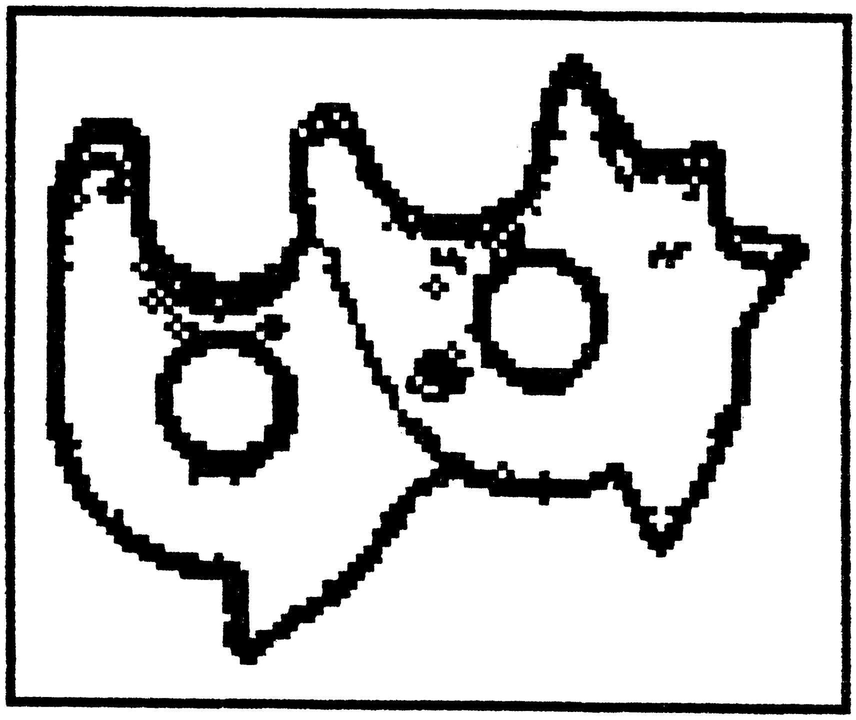
Picture 1



Picture 2



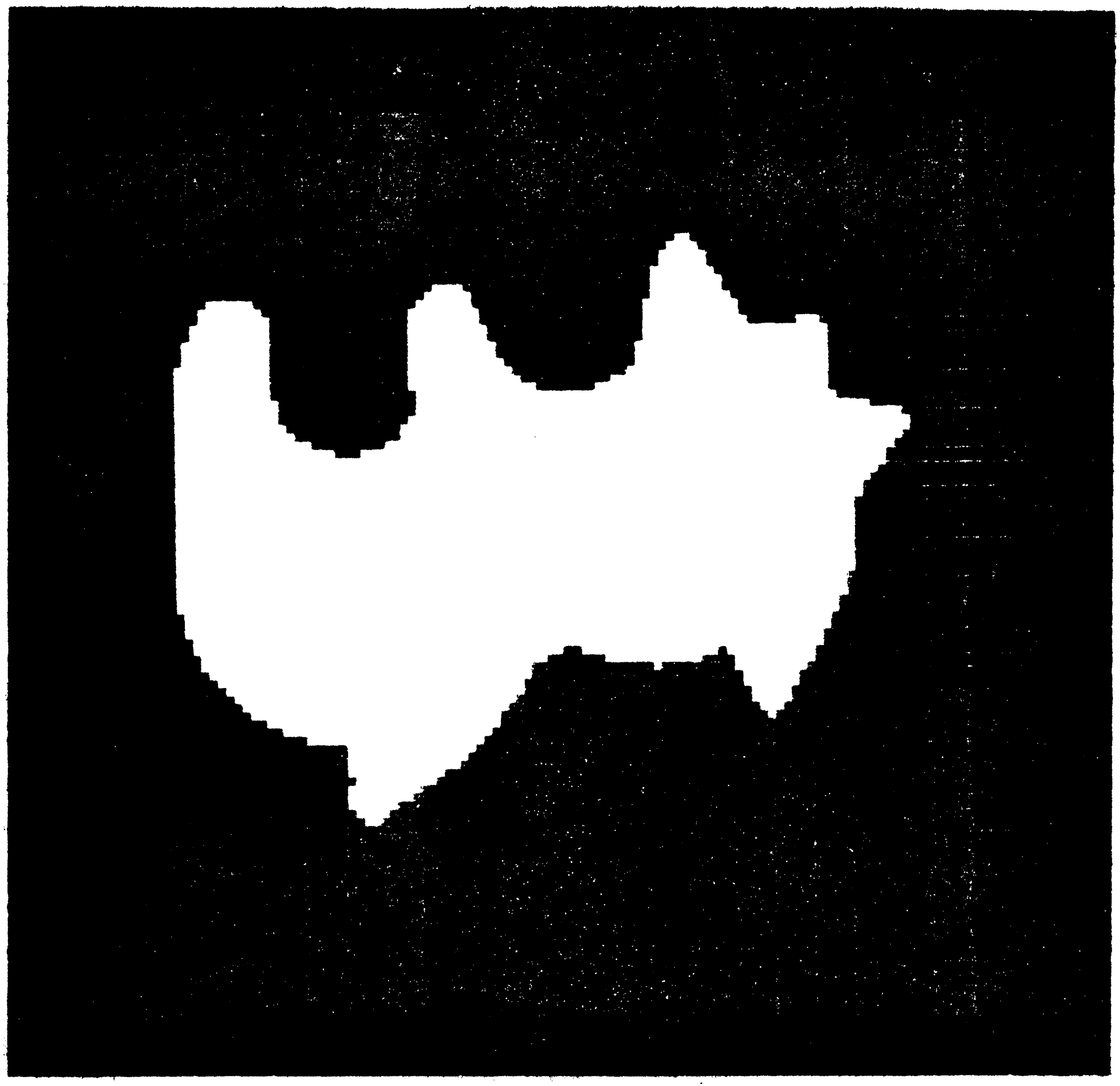
Picture 3

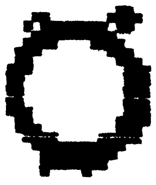


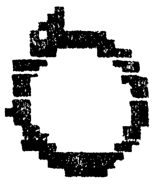
Picture 4

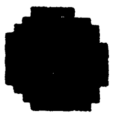
10. Appendix F: Labeled Images

This appendix shows the individual parts of the objects after they have been processed by the labeling stage.









11. Appendix G: Reconstructed images

This appendix shows the images after they are recognized and reconstructed.

Picture 2

Picture 4

12. References

[1] Agin, G. J. and Duda, R. O., "SRI Vision Research for Advanced Industrial Automation," *Artificial Intelligence Center, Stanford Research Institute*, Menlo Park, CA.

[2] Agin, G. J., "Basic Connectivity Analysis - DRAFT," "Run Length Coding", *Artificial Intelligence Center, Stanford Research Institute*, Menlo Park, CA, March 3, 1976.

[3] Ballard, D. H., Brown, C. M., Feldman, J. A., "An Approach to Knowledge - Directed Image Analysis," *Computer Vision Systems*, Academic Press, 1978.

[4] Bolles, R. C., "Robust Feature Matching through Maximal Cliques," *Proc. SPIE Technical Symposium on Image and Assembly*, Washington, D.C. (April 1979).

[5] Bolles, R. C., "Parts Acquisition using the SRI Vision Module," *IEEE Computer Society's Third International Computer Software and Applications Conference*, Chicago, Illinois, (5-8 November 1979).

[6] Bolles, R. C., "Locating Partially Visible Objects: The Local Feature Focus Method," *SRI Technical Notes 223*, Artificial Intelligence Center, SRI International, Menlo Park, CA.

[7] Birk, J., et al, *General Methods to Enable Robots with Vision to Acquire, Orient, and Transport Workpieces*, Fourth Report, 8/15/77 to 7/15/78, University of Rhode Island.

[8] Birk, J., et al, *General Methods to Enable Robots with Vision to Acquire, Orient, and Transport Workpieces*, Sixth Report, August 1980, University of Rhode Island.

[9] Rosenfeld, A., "Picture Processing by Computer," *Computing Surveys*, Vol. 1, No. 3, September, 1969.

[10] Rosenfeld, A., "Progress in Picture Processing: 1969-71," *Computing Surveys*, Vol. 5, No. 2, June, 1973.

[11] Shirai, Y., "Recognition of Real World Objects Using Edge Cue," *Computer Vision Systems*, Academic Press, 1978.

[12] Szwed, M.; Himlin, R. W.; Goris, D. R.; Stockton, D. W.; "Robot Vision System User's Guide," *Robotics Research Laboratory internal memo*, Department of Electrical and Computer Engineering, The University of Michigan, April 1981.

[13] Thompson, A. M., "Camera Geometry for Robot Vision," *Robotics Age*, March/April 1981.