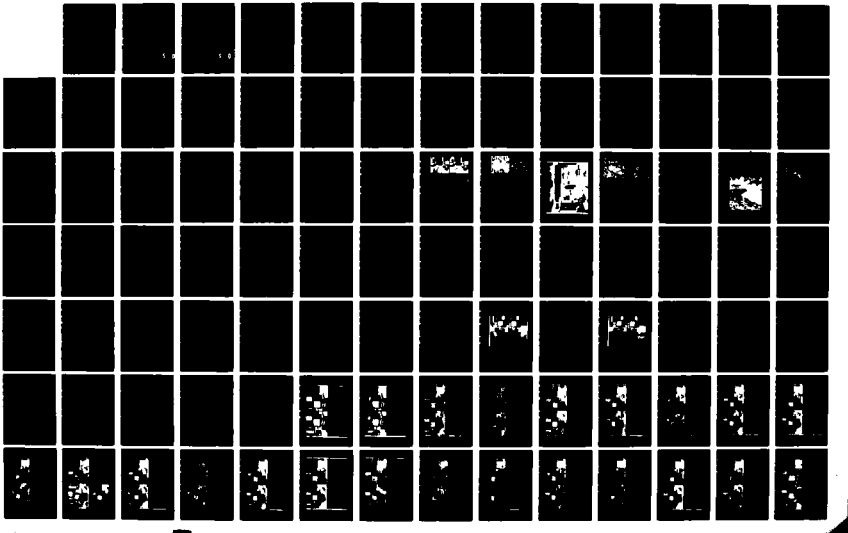
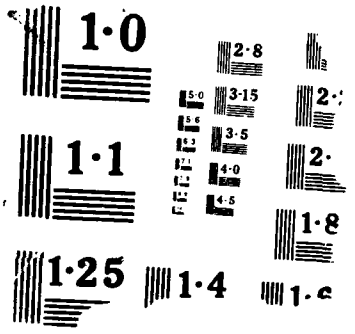
AD-A193 120 THREE-DIMENSIONAL SCENE ANALYSIS USING STERO BASED IMAGING(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING R E ROBERTS 09 DEC 87 1/2

UNCLASSIFIED AFIT/GE/ENG/87D-54 F/G 17/7 NL




1·0
1·1
1·25
1·4
2·8
3·15
3·5
4·0
4·5
5·0
2·
1·8

THREE-DIMENSIONAL SCENE ANALYSIS USING STEREO BASED IMAGING

THESIS

Richard E. Roberts
Captain, USAF

AFIT/GE/ENG/87D-54

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/87D-54

THREE-DIMENSIONAL SCENE ANALYSIS

USING STEREO BASED IMAGING

THESIS

Richard E. Roberts
Captain, USAF

AFIT/GE/ENG/87D-54

Approved for public release; distribution unlimited

AFIT/GE/ENG/87D-54

THREE-DIMENSIONAL SCENE ANALYSIS USING STEREO BASED IMAGING

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fufillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Richard E. Roberts, B.S.
Captain, USAF

December 1987

Preface

The purpose of this thesis was to investigate the ability of computer vision systems to emulate some of the capabilities of human vision.

A special thanks is due to Dr. Matthew Kabrisky, my thesis advisor, for his guidance and motivation throughout the months of this research. He always took the time to answer my questions. Thanks also to Capt Steven K. Rogers and Maj Phil Amburn for their support in organizing my research and keeping me on track, and especially for their patience during the preparation of this report. Thanks are also due to Capt Larry Lambert for his invaluable aid in debugging my programs, and for his many suggestions on better methods to implement my algorithms.

Table of Contents

List of Figures

List of Tables

Abstract

This thesis presents a new method for using passive binocular vision to create a map of the top-view of a robot's environment. While numerous autonomous robot navigation systems exist, most attempt to match objects in each image by following edges or locating significant groups of edge pixels. The method described in this paper uses two cameras (aligned in parallel) to generate stereo images. Low level image features are extracted using a new non-linear production rule system, rather than a conventional filter design. The features are registered by matching correspondingly shaped regions of constant brightness levels in both images and the offsets are then computed. The use of heuristics to relieve the computational burden associated with low level image processing is unique; both in processing the images and in locating matching regions in the images. The feature extraction algorithm, the intermediate symbolic representations, and the application of these results to hierarchical structures common to context queuing systems are presented.

THREE-DIMENSIONAL SCENE ANALYSIS USING STEREO BASED IMAGING

I. Introduction

Background

The ability to emulate the capabilities of human vision is a major requirement for the autonomous operation of mobile robots. While a simple video data chain may be sufficient to enable manual remote control of certain classes of robots, autonomous operation clearly requires that the robot have the capability to make decisions based upon information obtained visually. A major problem faced by current robotic vision systems is the inability of the robot to recognize and avoid objects in its path. This problem is complicated by the following factors:

a. The inherent size limitation of a robot which constrains the size, type, and number of optical sensors that can be used.

b. Constraints on the size of the computing element the robot can contain, and the amount of power available for the computing element.

c. The "real time" speed at which analyzed data is required for determining the environment of the robot.

The Robot Vision Problem

The major problems encountered with current robotic vision systems lie not in obtaining the images from the optical sensor; rather, the problems lie in understanding and interpreting the information in the image. The optical capabilities of robots exceed their computational ability. This thesis will study one method of increasing the computational capabilities of autonomous robot vision. The problem this thesis will address is whether the binocular distance measuring capabilities of the human vision system can be emulated using a computer and two video cameras. The goal is to use stereo vision to solve the problem of object avoidance, without having to solve the more difficult problem of object identification.

The purpose of this research effort is threefold. The first part is to study the feasibility of segmenting an image into "useful" regions of constant brightness values using the discrete edge representations of the objects in the image. The second part is to develop algorithms capable of matching corresponding regions in both images that were created in the first part. The third part is to calculate the distances to the regions matched in the second part and generate a partial three-dimensional representation of the scene based upon the distance measurements. Accomplishing these three steps should provide enough information for

autonomous motion of the robot by enabling it to effectively compute a path through its environment.

Scope

Calculating distances using stereo vision techniques requires two views of the same scene. Acquiring these views can be accomplished using either two cameras, or a single camera which is moved between snapshots. This research will use two cameras, like the human vision system, except that both cameras will be locked into an approximately parallel alignment and will be manually focused for each image. Because distances can be accurately measured with either method, there is no advantage in using independent movement or automatic focusing of the cameras; but there are many disadvantages resulting from the complicated control algorithms that coordinated camera movement and automatic camera focusing require.

The two common methods of computing distances from stereo vision are passive and active ranging. Passive stereo ranging is similar to human vision in that all information in the scene is obtained from the natural reflected light from each object. Active stereo ranging on the other hand scans the scene with a bright spot of light identifying each object exactly [Pou86]. While active stereo ranging is capable of providing more accurate distance measurements, this thesis will concentrate on

passive stereo ranging because it better suits the specific application considered.

Research will also be limited to methods of locating objects and calculating distances to those objects. Actual recognition of what the objects in the scene are will not be discussed. Details of the commercial software programs which obtain and store images of the scene will also not be presented in this thesis; these programs are part of the Imaging Technology series 100 image processing library and are being treated as incidental equipment which will not be modified in this study.

Research Objectives

Neither edge detection nor stereo vision is a new research area of robotic vision. This research effort will study methods of combining useful features from each of these areas into a system that can accurately compute the distances from the optical sensors to each unknown object. The algorithms this research develops could be an important building block in constructing an object avoidance capability in autonomous robots.

The first part of this thesis will describe a new class of edging algorithms. Before the vision system can compute distances it must be able to identify the same point on an object in two images. The feasibility of segmenting the image using edging techniques to assist in identifying

unique points on each object will be studied. The boundary between two objects will normally result in an edge because of the discontinuity in brightness values of the objects. Edges will not only separate different objects, but will separate each object into regions of constant brightness values. Identifying parts of objects that have the same brightness values should be easier than identifying the entire object.

The second part of this research will focus on developing methods of matching corresponding regions in the images and computing distances. It will attempt to create a matrix of numbers that represent absolute distances from unknown objects to the optical sensors on the robot. These distance measurements will represent only the sides of the objects that are within the field of view of the robot. Distances are computed to regions of constant brightness which might be composed of pixels from an isolated object, or a grouping of objects that are indistinguishable because they have the same brightness value in the image.

The final part of this research will focus on creating a top-down viewpoint image from the distance matrix computed in the previous step. This image will represent a re-creation of the original scene, where the robot's position and the position of the edges of known objects will be shown in a top-down view. Each edge in the scene will not necessarily represent a single object in the original

images. Objects that visually overlap and have similar brightness values in the original images may be grouped into a single region, and will therefore be represented with a single distance line. Objects that lie directly behind another object in the original images, and cannot be seen by the cameras, obviously can not be shown at all in the top-down viewpoint image.

Approach to the Problem

The distance measurement problem can be broken down into sub-problems that can be solved separately. The steps this research will follow in solving each of these sub-problems are as follows:

1. Develop image processing software using a C compiler and the MicroVAX computer. This will entail writing C language programs capable of exercising the image processing programs already present on the MicroVAX. These programs are capable of obtaining multiple images from the camera and storing them as a 512 x 512 bit array.

2. Reconstruct and simplify the QVA edging algorithm developed previously by Holten [Hol85b].

3. Verify that the QVA results obtained on the MicroVAX are equivalent to the results Holten obtained previously on the Data-General [Hol85a ,Hol85b].

4. Study the feasibility of using portions of objects with contiguous brightness values to locate and match in the

two images. To calculate distances the location of a point in each region in both images must be accurately measured. This step must be able to verify that there is enough information contained in the regions shape and pixel brightness values to uniquely identify and locate it.

5. Once it is possible to identify the location of the regions in both images, expand the system to calculate the distances from the optical sensors to each region.

6. The final step is to create an image that represents a top-down viewpoint of the area within the field of view of the robot. This top-down view will accurately depict distances, but will not represent actual sizes of the objects. This is because distance calculations will only be possible to the edges, representing the sides of objects facing the robots cameras. The reverse sides of the objects are not visible to the cameras and therefore can not be shown in this top-down viewpoint.

Standards

Exact object identification is not required because the purpose of this thesis is to develop a method to allow an object avoidance path to be computed. The system does not need to know what an object is to avoid it.

A partial three-dimensional or top-down representation of the scene should be good enough to perform initial path planning. Subsequent updates can be performed after motion

begins. Distance measurements to objects close to the robot are more critical than distance measurements to objects far away; distant objects are not immediate obstacles and can be more accurately processed or updated as the traveling robot approaches them more closely.

Because this is a prototype system, the speed with which the top-down viewpoint is generated is not as critical as the distance measurement precision. For this study real-time results are not required. It is more important for the system to compute the distances correctly.

Material and Equipment Required

The hardware and software requirements of this thesis effort are as follows:

Hardware

- Two - GE TN2505A video cameras
- Tektronix 4632 video hard copy unit
- Digital MicroVAX computer system
- DeAnza systems color image display system
- Imaging Technology series 100 image processor
- Camera mounting bracket
- Tripod support for camera bracket

Software

- Imaging Technology image processing library
- C language compiler

This thesis required the construction of special brackets to hold both cameras. These brackets are capable of maintaining parallel alignment of the cameras. The tripod allows leveling and allow rotation of the cameras. All of the other hardware and software requirements of this thesis are standard signal processing laboratory equipment.

Organization of the Thesis

This chapter provided a brief discussion of the problems associated with developing an object avoidance capability for autonomous vehicles. The objectives of this research were presented along with the scope, standards, and the specific approach that will be followed to meet these objectives. The remainder of this thesis will provide details of how each sub-problem identified in the approach was accomplished and how the results of each step were attained.

Chapter 2 will provide a literature review of some of the well known edging techniques, current applications of stereo vision, and a brief description of the capabilities of two autonomous vehicle systems which incorporate vision systems for determining their motion. Chapter 3 will outline the capabilities of the QVA edging algorithm, and discuss why the QVA algorithm produces the cleanest and sharpest edges of all the edging algorithms available today. Chapter 4 will present the method used to calculate

distances, and will outline the algebra involved in these calculations. Chapter 5 will present the method developed for comparing, identifying, and locating unique objects in both images. This chapter will also show the results obtained when the distance algorithm is incorporated and the top-down distance measurements are calculated. Chapter 6 presents the results obtained during this thesis. Results of applying the QVA production rules to an image are presented along with the results of calculating distances to objects trigonometrically. Also included in Chapter 6 are results obtained from four image processing techniques that were studied to improve the smoothing capabilities of the QVA production rules. Chapter 7 will present the conclusions reached during this study on the capabilities of the distance measuring algorithm developed. Chapter 7 will also discuss recommendations for further study and enhancements that can to be incorporated to make the system more robust.

II. Literature Search

Justification

Edge detection of objects in a scene has been one of the most active fields in machine vision. One of the first paper describing edge detection was by Roberts in 1965. It showed how to extract edges of polyhedral objects from a digitized photograph stored on a memory drum [Hor86]. Blicher though, points out that the paramount obstacle to image understanding is that there still doesn't exist a reliable and practical segmentation method which divides an image into meaningful parts based on edges [Bli85].

One of the first steps in image processing is finding an object in a scene. The location and size of the object must uniquely identify it from any other object in the scene. Often the interesting events in a scene, such as the boundary between two objects, lead to discontinuities in image brightness values, which are usually referred to as as edges. Looking for edges of objects rather than entire objects is usually better because edges are not affected as much by noise in the scene.

Edge-finding techniques are useful in locating the boundaries of objects, but many factors can corrupt the results such as shadows on the objects in the original scene, or two objects with almost the same brightness values located next to each other [Hor86]. Edges don't just

separate one object from another object, they also separate each object into regions with the same brightness levels. Determining the position of these regions should be easier than locating the position of each entire object.

A projected path can be determined even if only partial recognition of each object is accomplished. The robot can determine the distance to the object even if it doesn't know what the object is. An obstacle avoidance path around objects in the scene can be calculated because the complete description of the object is not required.

Discussion of the Literature

James Holten developed an elegant algorithm for analyzing a scene and generating a new representation of the scene identifying the edges of objects. This algorithm, called the Queen Victoria Algorithm (QVA), uses only additions and subtractions in its production rules to generate the edges [Hol85b]. This allows fast computations without using an array processor. The QVA also smooths the edges in the scene. This smoothing gives a more defined edge to compute distances from and thus enables more accurate distance measurements [Kab87].

The sharpest edges in a scene are easy enough to find, but Blicher points out that the global picture may require knowledge of how all the local qualitative features of the image fit together. Using information on where the bumps,

rifts, ridges, dips, etc. are and how they interlock may be required to completely analyze the scene [Bli85].

The current prototype of the autonomous land vehicle the Army is developing utilizes only one camera. It uses a simple edging algorithm to find edges of roads because of the data processing speeds required. It can easily become confused if shadows cover the road. All distance calculations are accomplished using either sonar or a laser ranging system, rather than visually. Extensions to the vision system to incorporate 3-dimensional viewing is still in the prototype development stage [Aut85].

Two of the most advanced autonomous robots are at Stanford and CMU. The Stanford Cart and the CMU Rover both use vision algorithms developed by Hans Moravec. They plan movement through a scene using only information obtained from on-board TV systems. The CMU Rover was developed after the Stanford Cart and has more capabilities [Mor83].

Both of these autonomous robots perform local statistical correlation between "high interest" areas in the scene. These data are then used to construct a 2-dimensional model with circular regions surrounding each obstacle. The algorithms then determine the next direction of motion from an obstacle-avoiding path. The path selected needs only to prevent the vehicle from intersecting an object's circle while it is traversing towards some goal position [Mor83]. But these algorithms suffer because of the extensive

computer time required to generate the circular regions. The scene must be analyzed to determine where the obstacles are, then the robot is moved a small distance and a new view of the scene is then analyzed. Documentation published in 1985 stated that the Stanford Cart required about 15 min for each 0.75 meter movement and the CMU Rover required about 1 min for each 1.0 meter moved [Hol85b]. Information on the specific type and computational speed (flops) of the computers used on these two autonomous robots was not included in the article.

One common way to compute range distances is using the technique of binocular stereo vision. Poulos describes this technique and discusses the algebraic computations necessary to determine distances using two cameras [Pou86]. This technique requires measuring only two angles because the distance to the point is a function of the different angles and the separation distance between the two cameras. The angles can be directly calculated by the offset position of the point from the vertical centerline of the image [Pou86].

The distance between two objects is easier to calculate than the distance from the cameras to one particular object. The range calculations become more accurate the farther apart the cameras are placed, but this poses two problems. The first is the size limitations of the robot. The second is that excessively wide camera separation adversely affects

correlating objects in the two images because of the small size of the video image used [Pou86].

The key to binocular stereo vision, and the largest problem, is determining which point on an object in one image is the same point on that object in the other image [Hor86]. Errors in selecting corresponding points in the images will result in errors in the angles and thus errors in the distance calculations. By placing the two cameras close together variations in ambient lighting between images can be minimized thereby reducing error [Pou86].

Using the technique of binocular stereo vision is computationally intensive but represents an alternative method for determining the relative positions of objects without precise knowledge of what the objects are. This technique is somewhat error prone, but because of the large quantity of data points used in the calculations the cumulative errors in the range results should be small.

Correct placement of the cameras, along with using QVA production rules to provide clean edges, should enable accurate distance calculations. Some error is expected because all distances are computed for static scenes. No information is retained from previous sets of images. The primary advantage of the static technique is that no apriori knowledge of the object's size or color is required to calculate the distances. Another advantages is that it doesn't require large or complex equipment to implement.

III. Queen Victoria Algorithm

The first step in the distance calculating procedure is preparing the images for region matching. The image must be processed to emphasize brightness differences or edges. The QVA (Queen Victoria Algorithm) was selected to accomplish this for this thesis. This chapter describes the background of the QVA, and provides a detailed description of how feature vectors were created for an image, as well as how the pixel brightness values in the image were smoothed using QVA production rules which created crisp well defined edges.

QVA Background

The QVA algorithm was developed in 1985 by Capt James Holten to smooth the pixel brightness values in an image and provide clean crisp edges for image processing [Hol85b, Hol85b]. The major differences between QVA and current edge enhancing algorithms are outlined as follows:

1. The QVA algorithm is spatially non causal. The QVA algorithm is not a real time application, rather it is applied to an image stored in memory. This allows the QVA algorithm to look ahead and use "future" pixel values (pixels with higher indexes) to smooth present pixel values.
2. The QVA algorithm is constructed from production rules instead of a transfer function or an impulse

response. The algorithm has none of the realization limitations of a system that can be modeled with differential equations.

3. The QVA algorithm is heuristically modifiable and can change as a function of the data. The number of times the QVA algorithm is applied to an image can be a variable depending on the smoothing results required.

Purpose

The purpose of using the QVA algorithm in this thesis was to process input video to create smooth regions of constant brightness separated by sharp edges. The QVA algorithm was necessary to prepare the images for region identification because the shape of each region was determined from the inside out, and represented groups of pixels with the same brightness values. Edges in the image represent the boundaries of the regions and constrain the pixel flooding that occurs during the region identification procedure.

QVA Pixel Smoothing Technique

The QVA production rules were applied prior to initiating a comparison algorithm which matched the regions created by QVA in the images obtained from each camera. The QVA algorithm uses two processes to accomplish this smoothing. First, feature vectors are derived from linear

sequences of individual pixel brightness values in the image. Second, a set of heuristically derived formal production rules are applied to those feature vectors to determine where and how much smoothing is to be performed on the image. When applied to a single slice of video brightness pixels across a picture it derives the location of putative edge intervals and smooth intervals, as well as the grey level characterizations of the intervals. [Hol85a]

A feature vector is created for each row and column of the image from the differences between individual pixel brightness values in the image. Each pixel location is assigned one feature from a set of five types of features possible. Each entry in the feature vector is determined from the grey level brightness differences between successive pixels. Table 3.1 shows the five possible types of features and the difference criteria for determining each type, where dif = pixel[i]brightness - pixel[i-1]brightness.

Table 3.1 Criteria for Selecting Feature Vectors

CRITERIA	MEANING
dif $\leq$ -thresh	negative edge
-thresh < dif < 0	negative gradient, possible edge
dif = 0	smooth
0< dif < thresh	positive gradient, possible edge
thresh $\leq$ dif	positive edge

The difference measure used in this thesis was the brightness difference between successive pixels in the image. This is not the only difference that could have been used, only the easiest to implement. The five types of features listed in Table 3.1 represent the minimum set of features. The threshold value controls how much brightness smoothing will be accomplished during each iteration of the QVA algorithm. While it is possible to vary the threshold value while the algorithm is running, the threshold was held constant during this thesis study to reduce complexity.

Production rules are then applied to the feature vectors to remove all brightness gradients between edges and sharpen all edges in the image. The production rules modify the actual pixel brightness values in the image based on the feature vectors created from that image. (I.e., given a video system with 256 brightness levels and a threshold = 0, then the QVA algorithm will not change any of the pixel brightness values; while using threshold = 125 the QVA results would be an image containing pixels with brightness values of 0 or 126 only). Smoothing all gradients will result in the image containing large areas of constant brightness separated by well defined edges. The width of each edge will be minimized because of this smoothing process, which results in clean sharp edges. The number of features evaluated by the production rules at any one time is variable depending on the type of features encountered.

The QVA algorithm keeps track of all pixels from the last edge position until a new edge location is indicated by the feature vector. All pixels are saved in an evaluation window while the features are examined.

The window used in the QVA algorithm will expand until an edge feature is encountered or until the brightness of a gradient feature exceeds the brightness of the first feature in the window by the threshold. If the feature that closed the window is a gradient then the gradient is changed into an edge of the same sign. Then the algorithm will determine the actual edge location by examining the features in the window from the right edge of the window to the left, with the following production rules:

1. If the feature is a positive gradient with a positive edge to its right then the gradient is changed into a positive edge.

2. If the feature is a negative gradient with a negative edge to its right then the gradient is changed into a negative edge.

3. If the feature is a positive gradient with a negative edge to its right then the gradient is changed into a smooth feature.

4. If the feature is a negative gradient with a positive edge to its right then the gradient is changed into a smooth feature.

5. If the feature is a smooth then stop.

When a smooth feature is encountered or created then the algorithm will smooth all features from that position to the left edge of the window. Next the starting point of the window is reset to the next feature following the right edge

of the current window and the process is repeated. The QVA algorithm will continue this process until all entries in every feature vector has been evaluated, and only smooth features and edge features remain. [Hol87a] To specifically illustrate how the QVA production rules would select the placement of an edge and which pixels would be smoothed, an example is shown in Figure 3.1, which is representative of a single row of typical pixel brightness levels in an image.

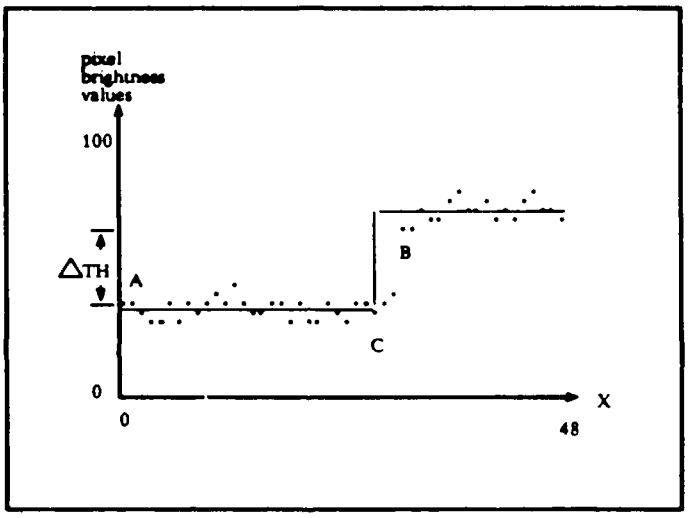


Figure 3.1 Example of QVA Edge Placement

The pixel brightness level at point "A" established the starting brightness level of a smooth section. The pixel brightness level at point "B" was the first pixel to exceed the threshold (TH). The starting position of the edge was located at point "C" though, because the pixels between points "B" and "C" were all positive gradients and point "B" was a positive edge (the pixel brightness value at point "B" was greater than point "A"). The smooth sections of the

graph represent the average brightness levels of all the pixels contained in each section.

The QVA algorithm allows the image to be processed iteratively until pixel brightness values stabilize to a relatively constant value between iterations. Each iteration of the QVA algorithm actually performs two passes over the image.

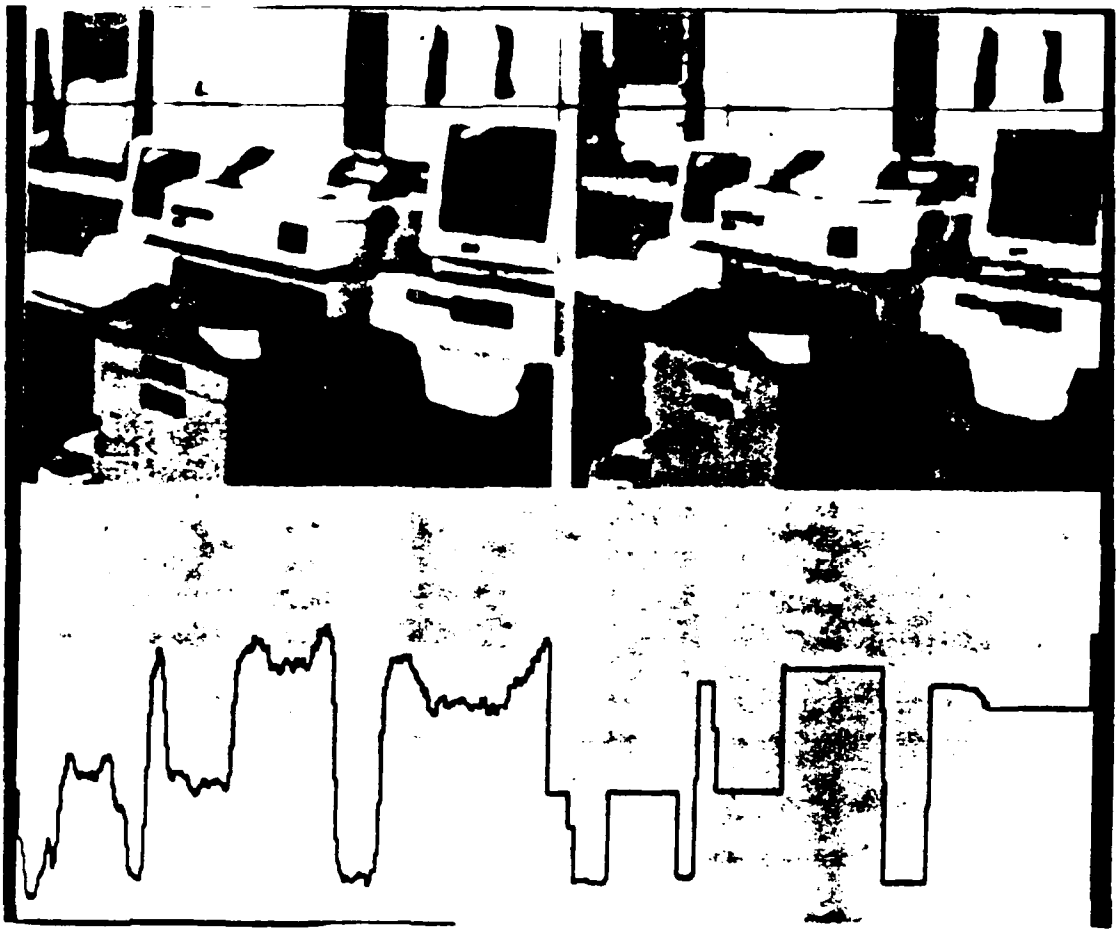
1. The image is processed from left to right. Feature vectors are created for each row and the production rules are applied to find all the edges and smooth all the pixel brightness values between the edges.

2. The image is processed from top to bottom. Feature vectors are created for each column and the production rules are applied to find all the edges and smooth all the pixel brightness values between the edges.

These two steps are repeated until the image brightness levels of the pixels are smoothed into large contiguous areas. The current criterion for smoothness is a subjective evaluation of the process. The QVA algorithm is repeated until the results shown on the monitor show very small changes between iterations. It is possible to define an objective criterion for smoothness and have the algorithm automatically check current image smoothness versus the criterion during iteration, but during this thesis effort all images were visually examined for their smoothness.

The actual implementation of the QVA smooths the pixel brightness values to the average brightness level of the object. When a threshold = 16 is selected it does not necessarily mean that only colors 0, 16, 32, 48, ..., 240, 256 will be displayed. It is possible that the average color of one area might be smoothed to, say, 12 during one QVA iteration. This allows QVA to provide more accurate information to the object comparison algorithm. If the brightness values were smoothed to a preselected level then information on the object would be lost and would probably result in additional errors in object identification. Smoothing pixel brightness to an average level might promote "bleeding" from one region into another, and require more iterations to settle into stable brightness regions.

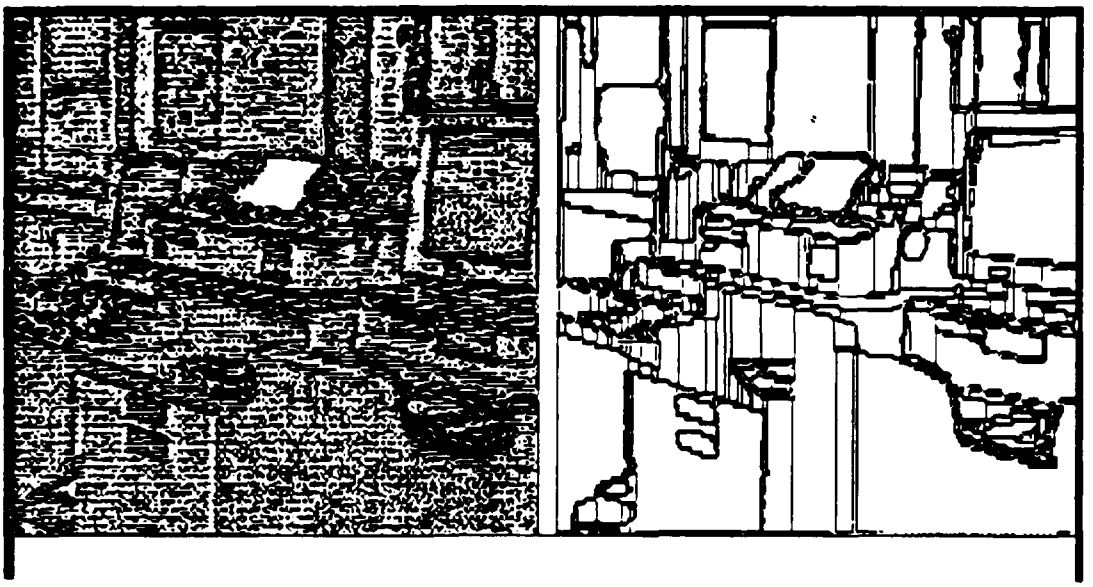
A typical QVA iteration result on one row of raw video image data is shown below to illustrate the effects of this process. The threshold was set at 16 and four iterations were performed (i.e. the image was processed left to right and top to bottom four times). Figure 3.2 (a) shows the original image and Figure 3.2 (b) shows the results after applying QVA. The pixel smoothing capability of QVA is demonstrated by displaying the actual pixel brightness values obtained from row 50 of both images below each image. The units of the graphs are pixel brightness values, with 0 at the bottom and brightness value 255 at the top.



(a) (b)

Figure 3.2 Example of Raw Data and Processed QVA Data.
(a) Raw Video Pixel Brightness Data,
(b) Processed QVA Pixel Brightness Data.

To further emphasize the smoothing capability of the QVA algorithm, a Laplaician edge detector was used against the unprocessed data of the original image and against a QVA processed image. Figure 3.3 shows the results. The Laplaician edge detector used was a 3X3 convolution with -1 at each kernel position except the center which was +8.



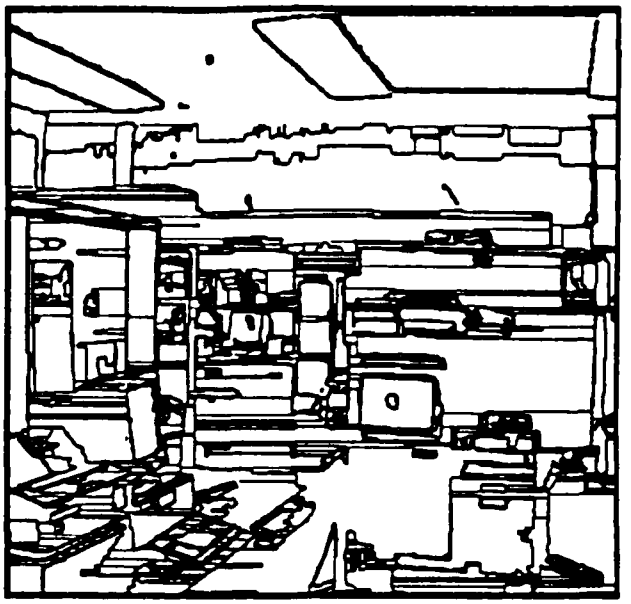
(a) (b)

Figure 3.3 Results of Applying Laplaician Edge Detector.
(a) Raw Data Prior to QVA,
(b) QVA Processed Image Data.

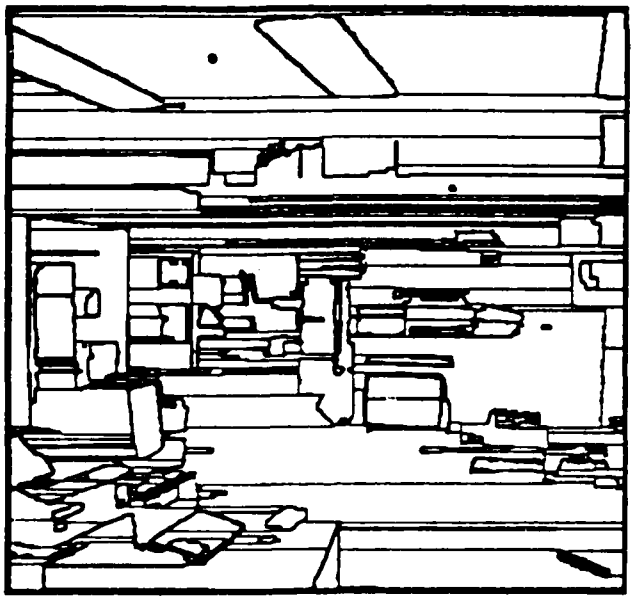
The criterion for determining which threshold to apply was based upon the types and sizes of the objects in the images. In general the larger the threshold selected, the larger the regions that QVA created. When the image contained man-made objects, thresholds above 32 tended to blend pixels from multiple objects into the same regions. Thresholds below 4 tended to eliminate video noise but didn't create regions of sufficient size to allow matching. Figure 3.5 demonstrates how the size of the regions created by the QVA process is dependent on the threshold selected. Figure 3.5 shows the results of using thresholds of 4, 16, 32, and 64 on the image shown in Figure 3.4.

Figure 3.4 Example Image for Comparing Threshold Size to Region Size Created by OVA

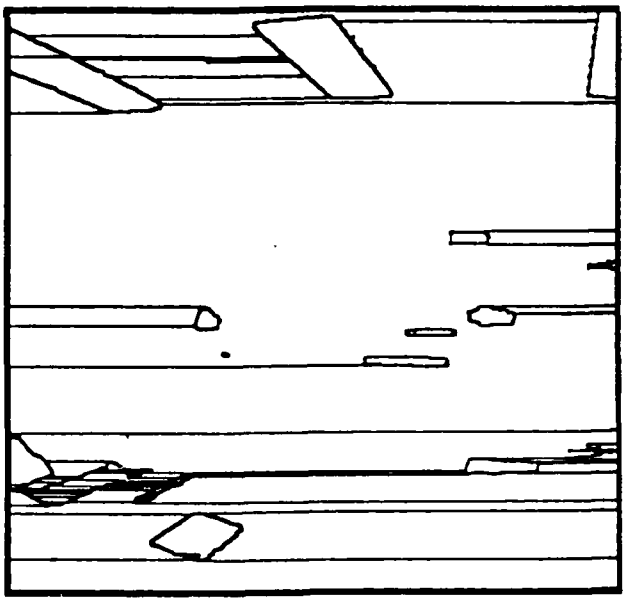
THRESHOLD = 4 ITERATION = 10



THRESHOLD = 16 ITERATION = 10



THRESHOLD = 32 ITERATION = 10



THRESHOLD = 64 ITERATION = 10

Figure 3.5 Region Size Versus Threshold Selected

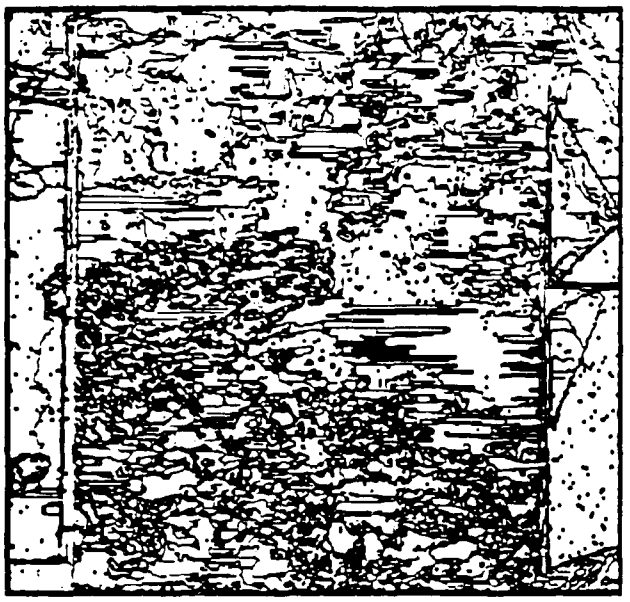
It was expected that QVA would work better in a man-made environment than in a natural environment. Man-made environments usually have sharp brightness edges between objects, and most of the objects have nearly constant pixel brightness values with brightness gradients caused primarily by lighting gradations. Whereas, natural environments which consist of trees, shrubs, and terrain details contain almost no sharp brightness edges and a plethora of pixel brightness values in many small regions.

To demonstrate how ineffective QVA production rules can be at creating regions of constant brightness in natural environment images, the QVA algorithm was applied with different thresholds to the image shown in Figure 3.6. The brightness edges of the four images shown in Figure 3.7 clearly demonstrate the inability of QVA create regions composed of pixels from single objects. When a small threshold was selected the image contains too many regions to be useful, and when larger thresholds are selected the pixels from multiple objects are combined into the same regions. Natural images tend to become confused or gratuitously complex when subjected to QVA. The only type of "feature" that QVA identifies seems to be texture.

QVA works best on images composed from man-made objects also because the colors (or brightness values) of most of these objects is grouped into large contiguous areas. The processing format of QVA also fits man-made objects which

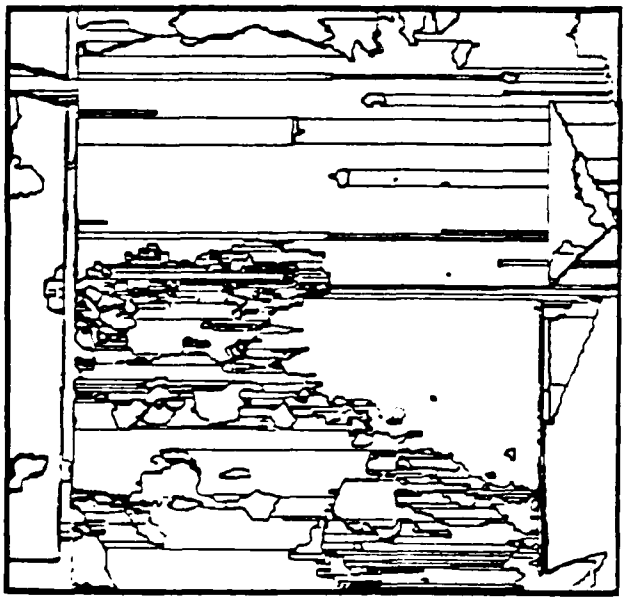
have many linear edges separating their colors. QVA takes advantage of long brightness gradients when smoothing. For this same reason QVA is inappropriate for most natural environment images, where very few of the objects have large contiguous brightness areas. A 3X3 convolution that performs local brightness smoothing would in most cases smooth a natural image into regions better.

Figure 3.6 Natural Environment Example Image

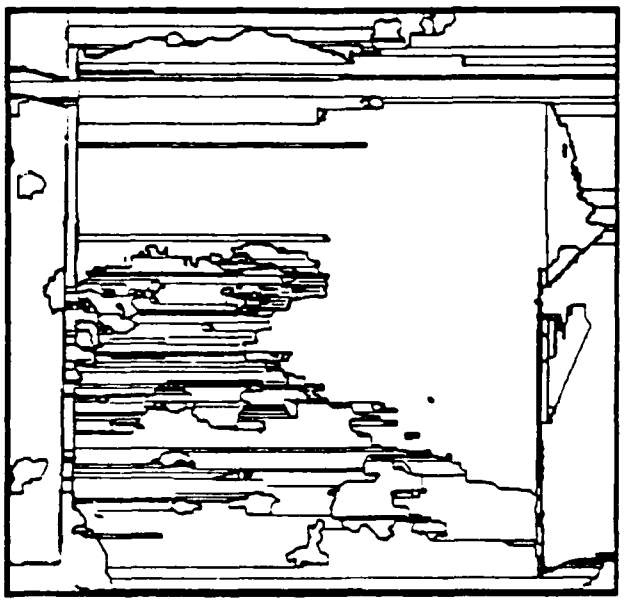


THRESHOLD = 4

THRESHOLD = 8



THRESHOLD = 12



THRESHOLD = 16

Figure 3.7 Results of Applying QVA to a Natural Environment
QVA threshold = 10 for all images

IV. Stereo Vision Distance Algorithm

This chapter will describe how distances were calculated from objects in a scene to the cameras. The trigonometric relationship between the position of the objects in the scene and the camera separation distance are presented along with examples detailing the actual implementation.

Introduction

"The principles of stereo vision for three-dimensional data acquisition are well-known and can be applied to the problem of an autonomous robot vehicle. Coincidental points in the two images are located and then the location of that point in a three-dimensional space can be calculated using the offset of the points and knowledge of the camera positions and geometry." [Hol paper]

Description of the Algebra

This technique requires two cameras viewing the same scene and mounted parallel to each other on the same horizontal plane. The distance Z from the cameras to an object P can then be calculated in terms of the angles $\Theta 1$ and $\Theta 2$, and the camera separation distance b [Pou86]. Using Figure 4.1 the law of sines gives

$$\frac{R1}{\sin \Theta 1} = \frac{R2}{\sin \Theta 2} = \frac{b}{\sin(180 - \Theta 1 - \Theta 2)}$$

where

$$Z = R1\sin(\theta 1) = R2\sin(\theta 2)$$

then substituting and solving for the distance Z gives

$$Z = \frac{b \sin(\theta 1) \sin(\theta 2)}{\sin(180 - \theta 1 - \theta 2)} \qquad \text{(Eq 1)}$$

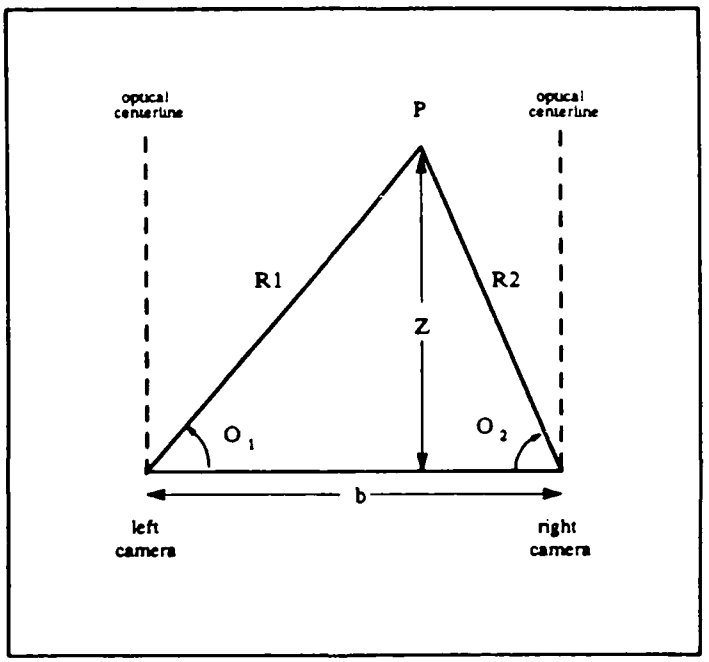


Figure 4.1 Graphical representation of the camera separation distances to a unique position.

The angles θ1 and θ2 in Figure 4.1 can be calculated from the pixel displacement from the image centerline position to the point P. Each image is 250 pixels wide, therefore, the centerline for each is at position 125. Figure 4.2 shows a graphical representation of this displacement from centerline.

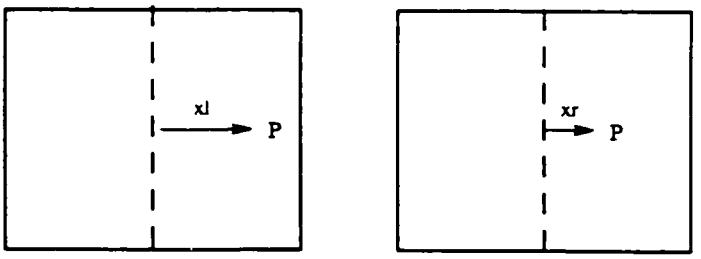


Figure 4.2 Example of Pixel Displacement from Centerline
(a) Left camera image (b) Right camera image

To convert from pixel displacement to degrees use the conversion factor 1.592deg = 22 pixels (see section Actual Implementation Details). Combining these factors, then θ1 and θ2 in Figure 4.1 can be determined as follows:

$$\theta 1 = 90 - (x\_left - 125) * (1.592 / 22) \qquad \text{(Eq 2)}$$

$$\theta 2 = 90 - (125 - x\_right) * (1.592 / 22) \qquad \text{(Eq 3)}$$

Eq 2 and Eq 3 are only correct when the point lies between the cameras as shown in Figure 4.1. If the point lies to the left of the left camera, as shown in Figure 4.3, then θ2 in Eq 3 is correct but θ1 in Eq 2 needs to be changed as follows:

$$\theta 1 = 90 + (125 - x\_left) * (1.592 / 22) \qquad \text{(Eq 4)}$$

If the point lies to the right of the right camera, as shown in Figure 4.4 , then θ1 in Eq 2 is correct but θ2 in Eq 3 needs to be changed as follows:

$$\theta 2 = 90 + (x\_right - 125) * (1.592 / 22) \qquad \text{(Eq 5)}$$

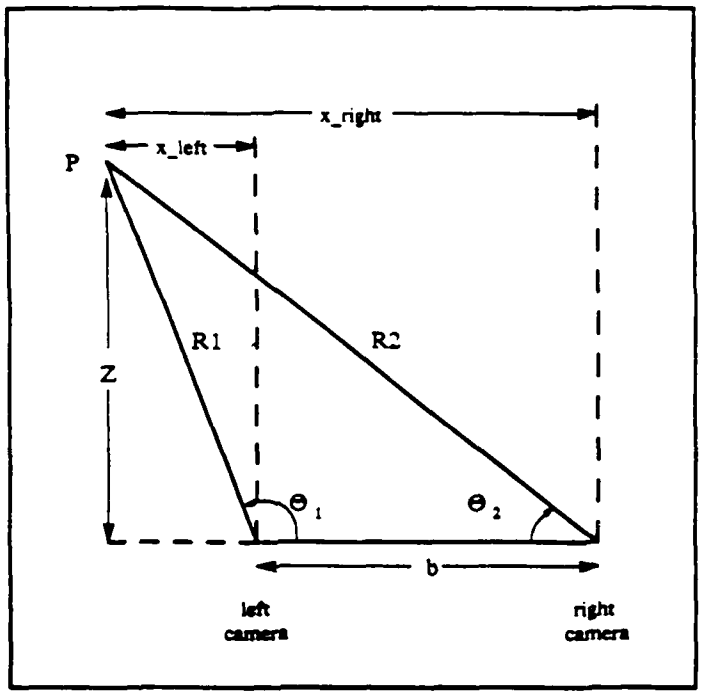


Figure 4.3 Target Point Location Lies to the Left of the Left Camera Optical Centerline

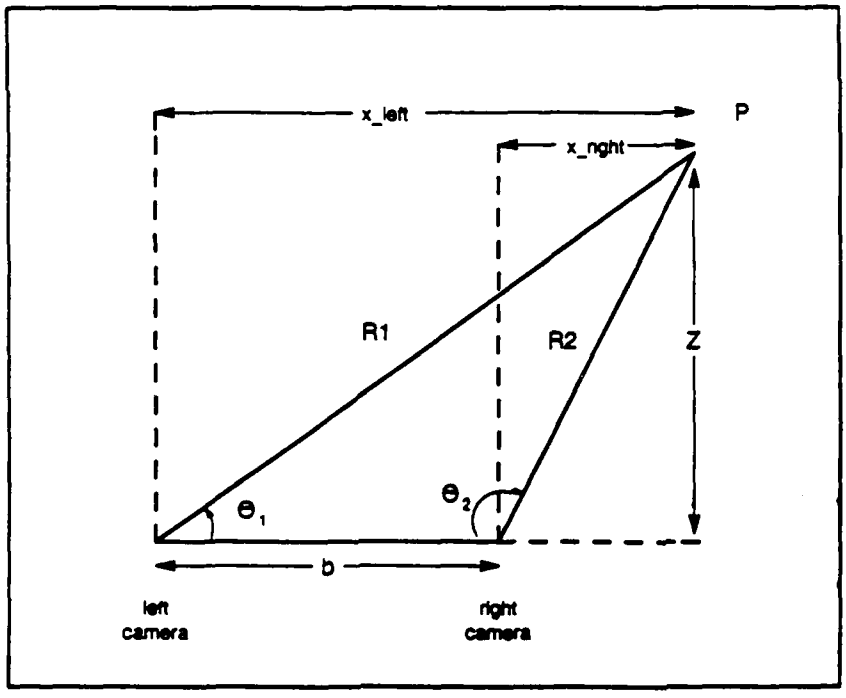


Figure 4.4 Target Point Location Lies to the Right of the Right Camera Optical Centerline

Actual Implementation Description

To determine the ratio between pixel displacement on the video monitor and number of inches in the actual image a simple procedure was used. The following steps essentially outline this procedure:

1. A target was placed 6 feet from the camera. The target was a piece of paper with a 2 inch grid on it.

2. The target was digitized and displayed on the video monitor.

3. Another grid was drawn on the monitor superimposed on the target grid pattern. Different size grids were drawn until one exactly matched the target grid.

It required a grid 22 pixels wide to cover the 2 inch target grid on the monitor. Using Figure 4.5 the ratio between pixel displacement on the monitor and the angle θ can easily be determined as follows:

Given $x = 2 \text{ inches} = 22 \text{ pixels}$

Then $\theta = 90 \text{ deg} - \arcsin(2\text{in}/72\text{in})$

Yielding $\theta = 1.59 \text{ deg}$

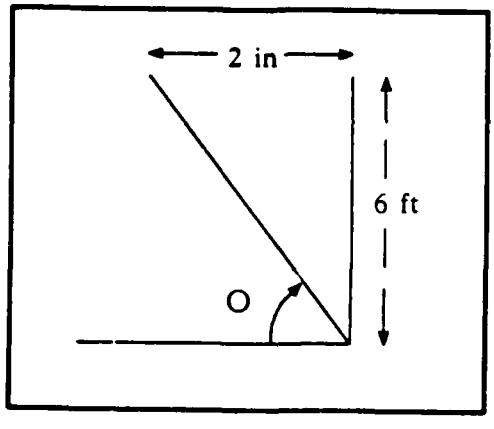


Figure 4.5 Relationship Between Pixel Displacement and Camera Viewing angle.

The conversion factor of 22 pixels for every 1.59 degrees is assumed to be constant for the width of the scene. Every object that lies on the same camera viewing angle will lie on the same vertical line in the video display. If vertical lines were drawn on the monitor every 138 pixels they would represent vertical slices of data from the scene taken every 10 degrees. Because the actual images used were only 250 pixels wide, they represent the area plus or minus 9 degrees from the optical centerline. This narrow image width allows us to ignore the non-linearity caused when the far edges of the image are digitized onto the video monitor.

Figure 4.6 shows an example with two points P1 and P2 that lie on the same viewing angle from the left camera, but lie on two different viewing angles from the right camera. The separation distance of the cameras (b) and the viewing angles of each camera from horizontal (θl, and $\theta r1$ or $\theta r2$) represent unique triangles, and the distance to each point (P1 or P2) is simply the height of each respective triangle. Because the ratio between the number of pixels a point lies from the optical centerline and the camera viewing angle was determined empirically, the algorithm probably contains a small error which will slightly effect the final distance calculations. Additionally, a small error in the distance measurements can be attributed to the internal alignment of the camera optics. The camera frames were mounted parallel, but the actual optical alignment was not verified.

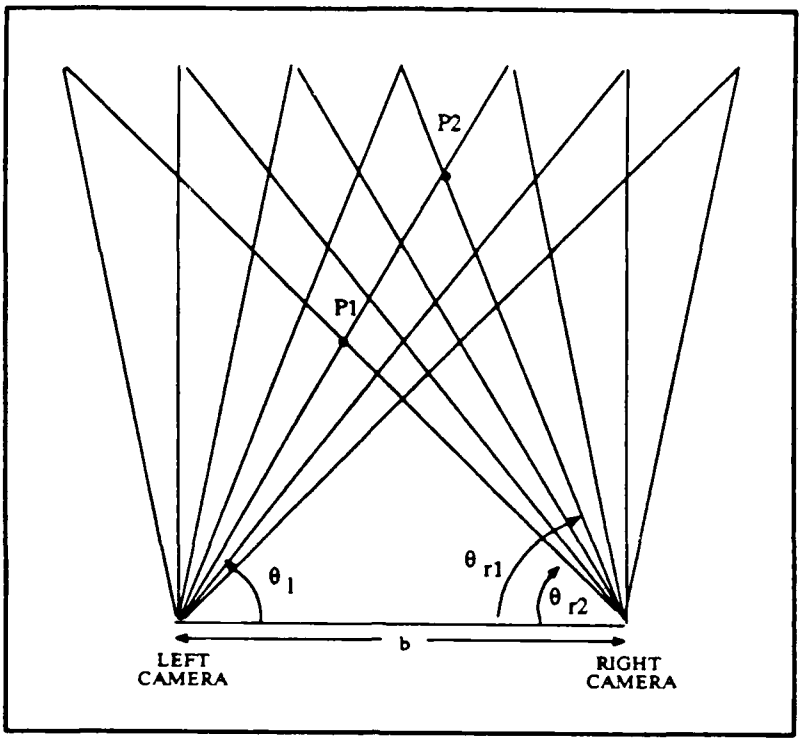


Figure 4.6 Example of Camera Viewing Angles

Description of the Distance Results Produced

The final distance results are displayed on the monitor with each distance to a region represented as a single line. An example of left and right image region matching is shown in Figure 4.7. Rectangles encompass the smooth pixel areas occupied by each region. For simplicity the actual image pixel data are not shown. In this example regions 1, 2, and 4 were matched and region 3 was unmatched. The distances to regions 1, 2, and 4 are depicted directly below the regions to which they correspond in the right image. The actual distance to each region in inches corresponds to vertical

pixel displacement on the monitor from the bottom of the screen up to each region distance line. The numbers to the left of the distance lines allows the user to obtain approximate distances quickly.

The area of each region matched is assumed to be perpendicular to the viewpoint of the camera. To be able to determine actual distances for every pixel, the program would need to be able to recognize a complete object not just a portion of it that is the same color. This thesis does not attempt a perspective measurement, which would require augmenting the current programs to perform complete pattern recognition.

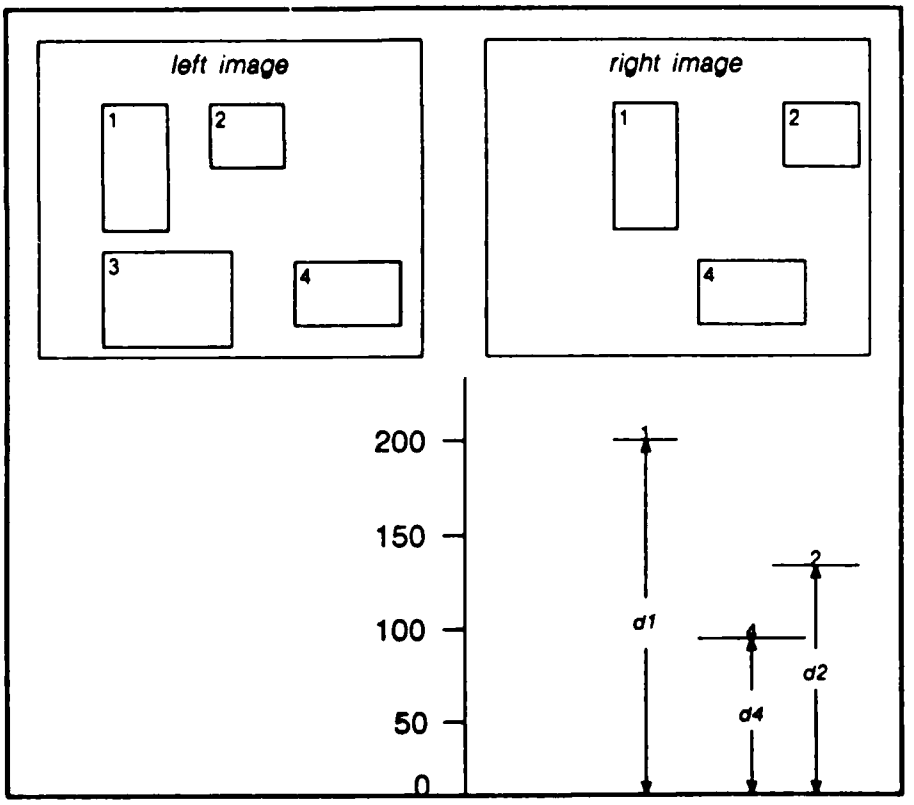


Figure 4.7 Example of Final Distance Results

V. Region Matching Technique

Accurate region matching is the most critical step in calculating the distance. A combination of image processing techniques was required to allow the algorithm to accurately match the regions of constant brightness in both images. Each of the image processing procedures used and the region matching techniques are separately described as follows:

Description of the Region Selection Process

The initial region selection process incorporated the following procedures to control how and where each region was identified and selected. Most of these procedures were empirically derived. These procedures were applied to the left image in this thesis to select the initial region.

1. The left image was used for initial region selection only because it was physically obtained first from the two cameras. The same results could be obtained using the right image to initially find the regions and the left image to find correlating region matches.

2. To limit the number of possible regions considered for matching a minimum size criterion was used. As will be seen, the minimum size was only used to eliminate regions too small to be significant and wasn't used to establish the actual size of the region. The software written to implement this part of the algorithm used four different minimum region sizes, 41x41, 31x31, 21x21, and 11x11. Each minimum

region size represented the smallest contiguous pixel area contained in the region. The first iteration examined the image searching for any region with at least 41x41 pixels with the same brightness value. If any pixel brightness values differ from the other pixels in the first area then the region search area is incremented over 1 column in the image and all the pixel brightness values are rechecked. After exhaustively searching for all the 41x41 size regions, the size of the minimum contiguous area of the region was reduced to 31x31 and the image was exhaustively searched again. This was repeated for 21x21 and finally 11x11. By constraining the size of the regions, small regions can be restricted from matching consideration which would slow down the operating speed. Regions smaller than 11x11 were considered too small to accurately match and likely to produce redundant results.

3. Once a candidate region meets a minimum size criterion, a graphics region flooding subroutine is instantiated to identify the entire region (i.e., a region is a group of contiguous pixels having the same brightness value). The starting coordinate of the flooding routine was set to the top left corner of the minimum size region identified in step 2. The region flood command used was from a ITEX100 graphics subroutine, and acts like a virus that expands from the inside out. Given a seed coordinate it will change all pixels of one brightness level into another brightness level without modifying any other pixels.

If the right threshold is selected the image will have large regions of constant brightness level pixels. The flood command results in regions of irregular shape. This is helpful in matching regions because it was expected that the constant brightness regions that QVA would create would have approximately the same shape in both images.

4. A rectangle was constructed around each flooded region identified in the left image to provide a region search size for comparison in the right image. The rectangular area was treated as a window which controlled how many pixels were evaluated during each iteration.

5. The search area of the left image is constrained to the right portion of the image (from x=60 through x=250). This is because most regions left of column x=60 are not visible in right image. X=60 in not an absolute limit, but rather a compromise. Regions on objects closer than 4 feet might not be fully visible after restricting the image search area, but fewer overall region mismatches were obtained for the entire image.

6. The search algorithm starts with large regions (minimum size = 41 x 41 pixels) and reduces the minimum size of the regions after each search iteration. (i.e. the entire left image is searched for regions of minimum contiguous pixel area size = 41 x 41 then searches the entire left image again for regions with a minimum size = 31 x 31 and so on). This was implemented because large regions are assumed to be generally closer than smaller regions and thus may

visually cover or obscure regions farther away from the camera. Once the location of a region was established it restricted other region from occupying the same location.

7. After a region is selected from the left image the area that region occupied in the image is masked off so that the same region will not be selected again for matching.

Heuristic Search Constraints.

Accurately matching the region selected from the left image in the right image required constraints on where in the image possible matching regions could reside. These heuristic constraints were derived during the development of the matching algorithm, and are described as follows:

1. The search area of the right image was constrained to x=0 thru x=200. Regions beyond x=200 in the right image are not visible in the left image.

2. The search area of the right image was constrained to +/- 10 pixels up or down from the position of the region in the left image. This is because the cameras were mounted at roughly the same distance from the floor and the images they recorded were therefore roughly the same in the y direction.

3. When looking for large regions the algorithm will search from from left to right side of the right image. This is because large regions are usually close the the cameras and their position will vary greatly from the left image to the right image.

4. When looking for small regions the algorithm will search a relatively small horizontal area determined by the position of the region in the left image. This is because smaller regions are usually far away from the cameras and their position will not vary greatly from the left to the right image.

5. To speed up the search algorithm the program varied the size of the loop increment based upon the size of the region it was searching for. For small regions the search loop incremented by one. for medium sized regions the search loop incremented by two, and for large regions the search loop incremented by three. This speeded up the loop but didn't decrease the accuracy of the distance measurement noticeable.

6. Once a region is matched, the area it occupies in the right image in masked off and other regions are not permitted to occupy the same area. Because regions that are close to the camera (usually large regions) are displaced farther between left and right images than regions that are a greater distance away, it is possible for small regions to be visually obscured from one camera behind a larger region. This results in some of the smaller regions not being matched, but because the larger regions are considered more important (i.e. they are visually closer to the cameras); this was not considered an error.

7. Additional matching errors are sometimes caused because the location of the matched regions are not allowed

to overlap. The matching algorithm selects the best matching position it can, but because it is prevented from overlapping regions it will place the matching region next to the previously selected region. This sometimes results in a matching offset error which propagates into the distance calculations and results in range errors.

Features of QVA that Supported Region Matching.

1. QVA results were consistent. The pixel brightness smoothing performed by QVA on both images created regions similar in shape and pixel brightness values.

2. QVA created large regions of constant brightness levels surrounded by crisp edges. When the flood command was initiated it would grow until all pixels within the region were modified (i.e. all the pixels within the edges). The range of pixel brightness values in the original images was from 0 to 254. The flood command changed pixel brightness values to 255. The images could then be searched to locate the rectangular boundaries of the flooded region. Without QVA the area flood technique is worthless.

Summary of the Implementation Details

1. Both the left and right images are 250 x 250 pixels in size. This size was chosen because it allowed separation of the images on the monitor. The program can be rewritten to use larger sizes (i.e. 256x256 or 512x512), but they would require more time to execute.

2. The ITEX 100 pixel flooding subroutine requires exclusive access to the ITEX 100 image processing board during execution. Using images of 250 x 250 size allowed a duplicate left image to be shown on the monitor during the calculations. One image was used to store the original pixel data and the other was used to record the results of the pixel flooding. When the calculations were complete the pixel flooded image can be deleted from the monitor leaving only the left and right images and the distance results.

3. Each pixel is represented by a brightness value between 0 and 255 (8 bits).

4. Rectangles were drawn around each region identified in the left image and each region matched in the right image. Each rectangle in the left image is labeled with a unique number and corresponding matches in the right image are assigned the same number as in the left image.

5. Distance calculations are measured from vertically from the bottom of the monitor up to each line, rather than from a single point. Each pixel on the monitor corresponds to one inch in actual distance to that region.

6. The region matching algorithm uses two steps. The first attempts to match regions using only the shape of the region in the left image. If the algorithm cannot determine a "best match" region, (i.e. summation of pixel brightness values from one region are at least 50 lower than all the rest of the possible region matches), then all the pixel information in the rectangular region is used. This second

method adds error to the distance measurements because sometimes the extra information in the rectangular area prejudices the match to the wrong area in the right image.

VI. Results

The accuracy of the distance calculations is dependent on the ability of QVA to properly prepare the images and the accuracy of the algorithm in matching regions of constant brightness in both images. This chapter will present the results that can be obtained by applying the QVA production rules to an image with different thresholds and different numbers of QVA iterations. The accuracy of the distance calculations will also be presented along with an example of the entire distance calculating process applied to the same scene from multiple distances. This chapter concludes by outlining results obtained after applying various image processing techniques that were initiated prior to applying QVA in an effort improve the region matching accuracy.

QVA Production Rules on an Image

To demonstrate the strengths and weaknesses of the QVA algorithm, the image shown in Figure A-1 was processed using QVA with seven different iteration values and four different thresholds. (see Appendix A for the figures used to compile Table 6.1). Table 6.1 summarizes the results of all 28 tests, where each test evaluated the ability of the region matching algorithm to correctly find and match regions in both images. The results represent the number of regions located in the left image, the number of regions correctly matched in the right image, and the number of regions incorrectly matched in the right image.

Table 6.1 Comparison of Threshold Value and Iteration

Thresholds	# of QVA Iterations: 2	4	6	8	10	12	14
4	10 5 3	19 16 3	18 18 0	19 17 0	20 16 1	20 18 0	21 19 0
8	17 14 2	21 19 2	21 19 1	22 19 1	22 18 1	19 13 0	20 14 0
16	20 14 5	21 17 4	29 17 6	21 13 4	18 12 4	19 16 1	17 14 0
32	28 16 9	23 15 5	23 13 6	27 16 6	24 14 5	28 14 5	25 14 1

Each entry in Table 6.1 was determined using the following criteria:

# of regions identified in left image / # of regions correctly matched in right image / # of regions incorrectly matched in right image

The difference between correct and incorrect object matches was a subjective evaluation. A region matched within 10 or 12 pixels of its correct position was considered a correct match because distance measurements were usually within 10% of the actual measured distance to the object. Fewer objects were usually matched in the right image than were identified in the left image because the matching algorithm restricts regions from overlapping in the right image. If one region is slightly misaligned it may

prevent the matching of an adjacent region. This is not considered a matching error, provided that at least one part of each object viewed in the images is correctly matched. Restricting the placement of regions was incorporated into the algorithm because it reduced the number of matching errors due to close regions obscuring far away regions from the cameras view.

The results shown in Table 6.1 are the first step in incorporating an automatic method that can select the right sized threshold to process a given image. Combining an automatic threshold size evaluator with an image smoothness test would eliminate the need for the user to monitor the process. Due to time constraints this enhancement was not implemented.

When the QVA algorithm was being tested, for threshold and iteration effects on the accuracy of the region matching, the following results were observed:

1. Smaller thresholds tended to produce fewer errors at any given number of iterations when compared to large thresholds. The size of the objects identified for matching was usually smaller than obtained with larger thresholds. All the matching objects were composed only of parts of complete objects in the image. The flooded shapes of the objects used in matching were very irregular in shape and this irregularity probably contributed to the small matching errors obtained.

2. Large thresholds tended to select more objects in the left image for matching because of the size criterion, but there were more errors and many more iterations were required by QVA to smooth the image and reduce the number of errors. When the threshold was set to 32 many of the objects identified for matching in the left image were not matched at all in the right image; even though more objects were initially identified the actual number matched was roughly equivalent to the number attained with smaller thresholds.
3. Selecting thresholds of 2 proved too small to allow matching any objects. This threshold was too close to the video noise and QVA couldn't smooth the image sufficiently for matching.
4. Using a threshold of 64 distorted the brightness levels so severely that matching objects were usually composed of parts of many objects in the image. Brightness levels from one object were propagated across large areas of neighboring objects resulting in mutilated images and attempts to match objects was futile. If a match was attained it contained objects from many different distances, thus, the distance calculated to that region was not meaningful.

It was noted that after approximately 4 or 5 iterations the results of applying QVA to the image didn't produce a difference that could be visually discerned on the video

monitor; yet, as shown in Table 6.1, the results continued to change with additional QVA iterations.

Region Distance Calculations

To show how well the distances were trigonometrically calculated to each region matched in the images, the calculated results were compared to the actual measured distances to the same regions. These comparisons were performed twice; once using a threshold of 6, and the second time using a threshold of 12. The results obtained using a threshold equal to 6 are shown in Tables 6.2 and Figure 6.1, and the results obtained using a threshold equal to 12 are shown in Table 6.3 and Figure 6.2 Measured distances were measured from the center point between the cameras to each region. The computed distance results were obtained by applying the QVA production rules 12 times to both images.

In Figure 6.1 the distance to region 15 was considered invalid due to a matching error in locating the correct position of region 15 in both images. In Figure 6.2 the distances to regions 3 and 15 are considered gross errors. The small error percentages for the other regions shown in Tables 6.2 and 6.3 clearly demonstrate that the distance algorithm developed in this thesis is an acceptable starting point for a vision system for autonomous robots. When the QVA algorithm is applied to two parallel images with sufficient number of iterations, it produces images with large regions of constant pixel brightness values that can

be identified and accurately matched in both images by their shape and brightness values, thus allowing distances to be computed to each region pair matched.

Table 6.2 Comparison of Measured to Computed Distances QVA Threshold=6 and Iteration=12

object #	measured distance inches	calculated distance inches	difference inches	error percentage
1	99	94	5	5.0
2	119	114	5	4.2
3	114	113	1	0.9
4	122	116	6	4.9
5	128	116	12	9.3
6	113	121	8	7.1
7	101	111	10	9.9
10	133	141	8	8.0
11	113	113	0	0.0
13	113	127	14	12.4
14	98	90	8	8.2
15	133	48	85	63.9
16	98	90	8	8.2
17	97	89	8	8.2

Figure 6.1 Results of Applying QVA to an Image 12 Times With a Threshold of 6

Table 6.3 Comparison of Measured to Computed Distances
QVA Threshold=12 and Iteration=12

object #	measured distance inches	calculated distance inches	difference inches	error percentage
1	99	94	5	5.0
2	119	114	5	4.2
3	114	172	58	50.8
4	130	118	12	9.2
5	122	116	6	4.9
6	128	126	2	1.6
8	101	104	3	2.9
9	97	91	6	6.2
10	113	121	8	7.1
15	105	1017	912	868.6
18	98	90	8	8.2

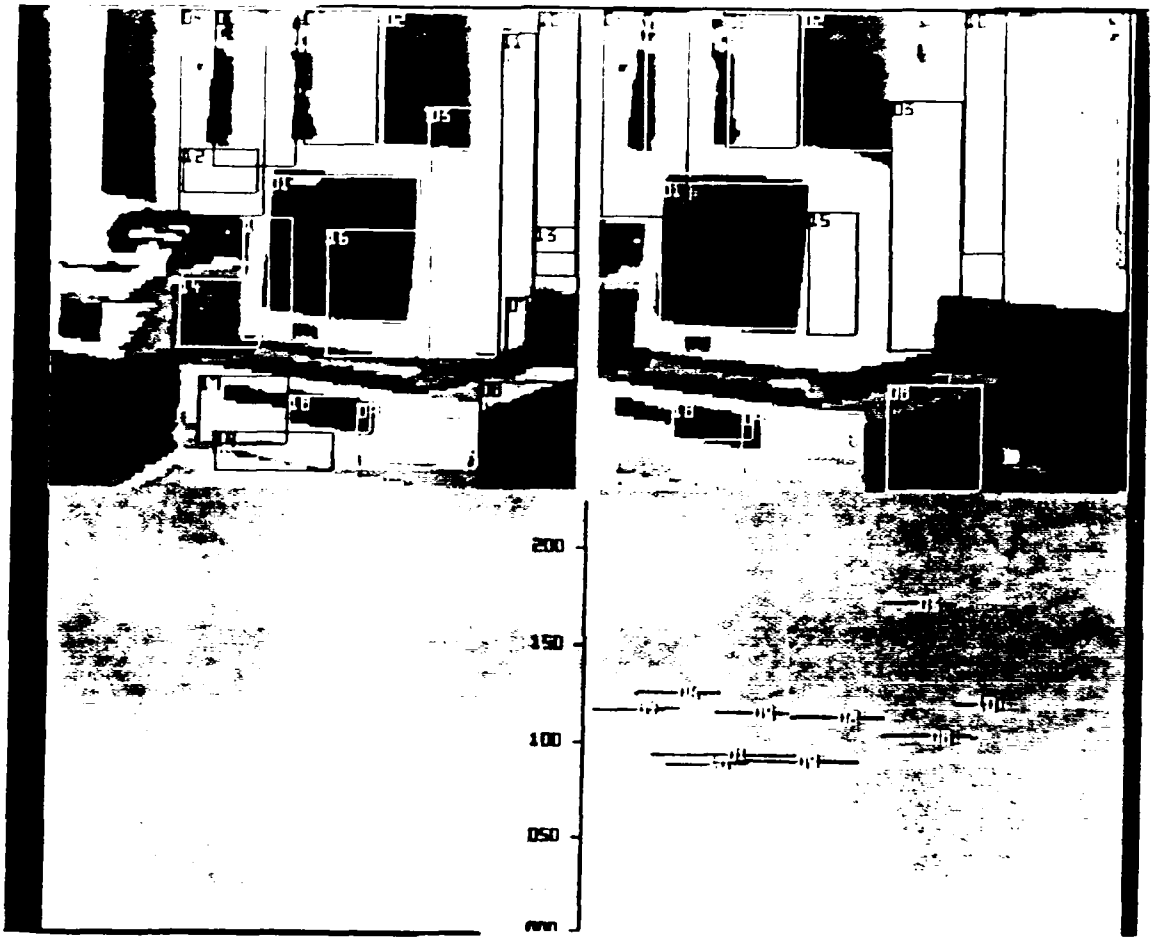


Figure 6.2 Results of Applying QVA to an Image 12 Times With a Threshold of 12

Image Preprocessing

Many of the errors resulting from inaccurate region matching were the result of the QVA production rules blending brightness values from multiple objects into a single region. This is sometimes referred to in this thesis as bleeding. This caused problems for the matching algorithm because it caused errors in the placement of the matching regions. Adding an image preprocessing step to usefully enhance the input images before initiating the QVA production rules would appear to be a logical addition; preprocessing would improve the results produced by QVA if the amount of bleeding could be reduced. Four types of preprocessing were tested for this purpose; they are listed below:

1. Edge enhancer
2. Low pass filtering
3. Normalizing the image brightness values
4. Averaging the image brightness values

Paradoxically the results of all four proved to be detrimental to the results generated by the QVA production rules. Fewer regions were matched and more of the matches were incorrect. A complete description of each process is presented below. It is not obvious why all of these methods failed. QVA is, of course, a highly non-linear procedure and this can easily result in failure of intuitively derived treatments.

1. Edge Enhancing

This process attempted to accentuate the brightness levels of pixels that formed the edge boundaries between objects in the image. It was proposed that this would reducing the bleeding or blending between regions with similar brightness values. (i.e. if the pixel brightness value of two objects are almost equal to the threshold then QVA will smooth some of the pixel brightness values of both objects into the same region). This causes confusion during region matching and usually results in matching errors. If the bleeding were reduced then matching errors would also be reduced.

The edge enhancer used in this thesis used a 12 by 12 convolution to modify the center 4 by 4 pixels. The average brightness values of all 144 pixels is subtracted off from each of the four center pixels. Then their resultant brightness values are multiplied by 5 and added back to the average. This should emphasize pixels whose brightness levels vary significantly from their neighboring pixels (i.e. possible edge pixels). [Fre87]

2. Low Pass Filtering

This process was initiated after applying the edge enhancing step. It was proposed that this would reduce the video input noise in the image and also reduce the effects of the edge enhancing which enhanced isolated pixels brightness values even though they were not part of an edge.

A 3 by 3 convolution was used to blur the image which tends to eliminate localized noise. The kernel used was:

```
1 2 1
2 4 2
1 2 1
```

The results of applying this convolution to the image were then scaled by 16 to normalize the pixel brightness values back to a value between 0 and 255. [Ite86]

3. Brightness Value Normalization

This process attempted to normalize the pixel brightness values of the objects. It was proposed that this would stop or reduce the bleeding of brightness values by spreading out the pixel brightness values to use the full scale from 0 to 255 available. Two objects whose original brightness levels were different by the threshold, would now be enough different so that bleeding would not occur and the number of matching errors would decrease. Because both cameras have the same focal length and apperature the images produced by them should contain pixels with almost exactly the same brightness levels.

4. Brightness Value Averaging

This process attempted to average the brightness values of each object in the image. The median value of a variable size convolution was obtained. The median was then subtracted from each of the pixels in the convolution area. Finally the results are normalized to values between 0 and 255. This process was attempted twice, using 10 by 10 and 16 by 16 sized convolutions.

VII. Conclusion and Recommendations

Conclusions

This thesis demonstrated the feasibility of using a computer and two video cameras to perform automatic range finding through binocular disparity. Applying QVA production rules and a graphical region flooding technique enable the segmentation of the images obtained by the cameras into regions containing pixels with the same brightness values and the matching of those regions in both images. Once the regions were matched it was then possible to calculate trigonometrically the distance from the cameras to each region and then display the results on a video monitor.

The shortcomings of this ranging system lie in the computationally intensive algorithms used to fully process each pair of images and in the user interaction required to control the number of iterations of the QVA algorithm applied to each image.

The horizontal position of the cameras was constrained such that each pair of images obtained were in the same horizontal reference plane (i.e., objects in both images were located at approximately the same vertical position and only the horizontal position was different). This allowed the area searched for each region to be vertically constrained; thereby accelerating the region matching portion of the algorithm. If the horizontal alignment of

the cameras is not constrained then the search area must be expanded vertically and the time required to match regions will grow accordingly.

Recommendations for Further Study

1. Modify the algorithm to allow the cameras to be independently aligned toward a particular object rather than being always aligned parallel to each other. This would require accurately measuring the alignment of each camera. Distances would be based upon the object the cameras were aligned toward rather than on the location of the cameras.
2. Modify the QVA iteration process to incorporate an internal pixel smoothness test. This could be based upon the relative pixel brightness value change between consecutive iterations or perhaps a histogram of the original pixel brightness values. This would enable the QVA algorithm to generate an acceptable smoothed image autonomously and without performing excessive and unproductive smoothing.
3. Develop an image processing algorithm that can be applied before QVA to reduce the pixel brightness bleeding encountered during the brightness smoothing. If every region is totally composed of parts of the original objects in the scene and not part of multiple objects, then fewer incorrect matches should result.
4. Develop a better region matching technique. The method used in this thesis performed an exhaustive and time

consuming pixel brightness value comparison over the entire area of every potential matching region, and then selected the region with the lowest difference in pixel brightness values over the regions entire area as the match.

4a. Enhance the matching algorithm by expanding the brightness input from a monochromatic greyscale to incorporate color. This could be accomplished by using a color camera or adding colored filter lenses to the existing black and white cameras and taking sequential views.

4b. Enhance the matching algorithm by using the texture of the regions in addition to their color or brightness values in the calculations. Texture is indicated by gratuitous patterns in the image that correlate with the boundaries of objects.

5. Develop a more effective difference criterion for creating the feature vectors before applying QVA production rules. Rather than using only successive pixels the difference could be an average of many pixels, both preceding and following a given pixel.

6. Expand the number of feature types associated with each pixel. The five types used in this thesis represent only the minimum set. Using multiple types of gradients could allow more accurate placement of edges in the image.

7. Adjust adaptively the minimum sizes of the regions used. It may be possible to assign the minimum size based upon the type of image to be processed. The speed of the distance algorithm might be significantly improved if fewer sizes

were used because each different size requires a complete pass over the image. Masking of previously matched regions provides only moderate speed up.

8. Transfer the QVA algorithm to a parallel processing machine (i.e. iPSC hypercube). The QVA algorithm lends itself well to a parallel processing format. The creation of feature vectors and the application of the production rules on each row or column of pixels in the image can be performed independently. Incorporating the algorithm on a parallel processing machine would eliminate the double loops currently used and allow "near real-time" image processing with mini-sized computers.

9. Incorporate an algorithm to determine the optimum size of the regions and number of iterations necessary to prepare the images for region matching. This would require duplicating Table 6.1 for many different images and developing heuristic rules for each type of image.

APPENDIX A

Example Images Used In The Thesis

This appendix containes copies of the images used to produce Table 3.2. The origional image in a raw data state in shown on page 64, and all other images in this appendix were created by applying the QVA production rules to this image.

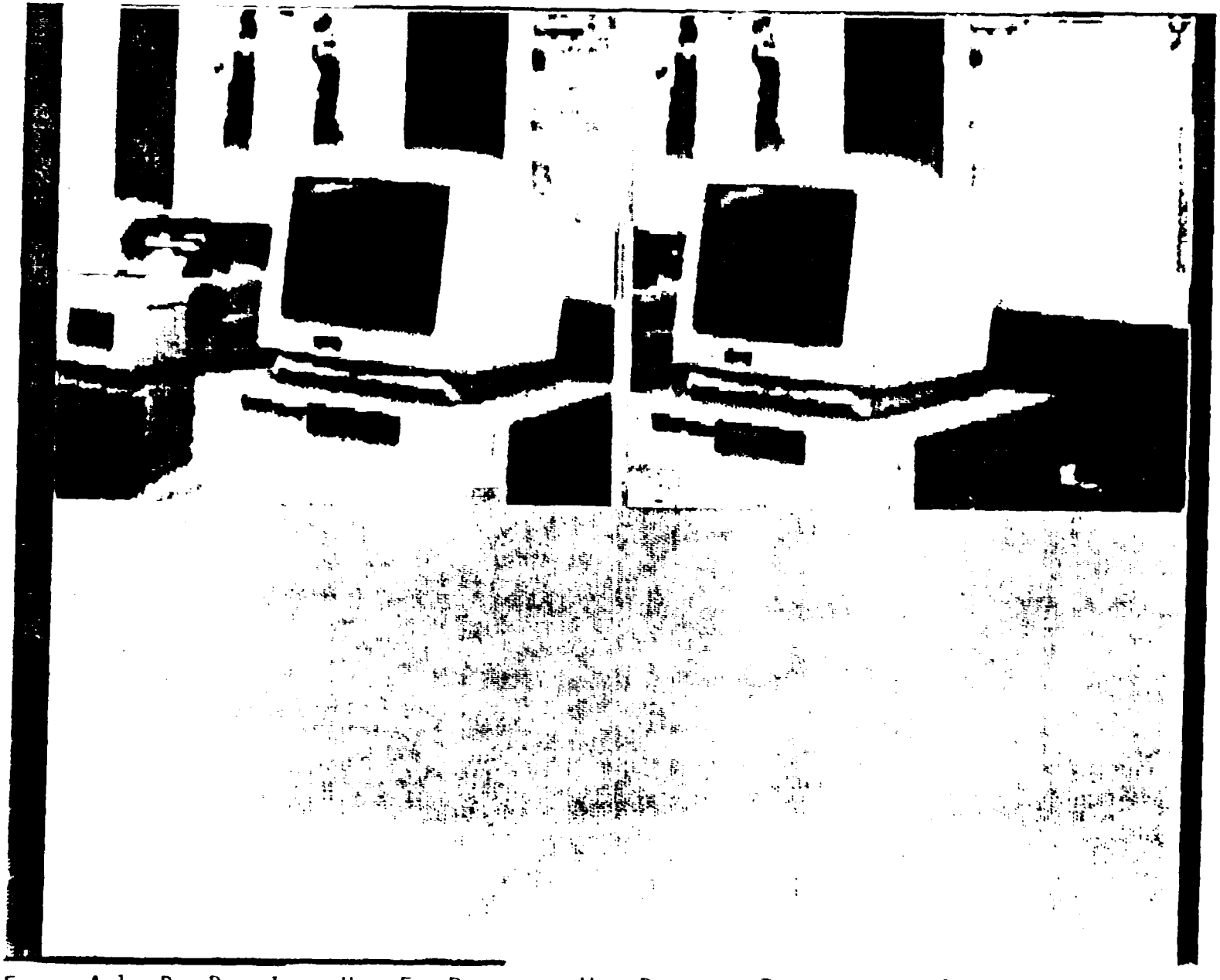


FIGURE A-1 RAW DATA IMAGE USED FOR PROCESSING WITH DIFFERENT THRESHOLDS AND ITERATIONS

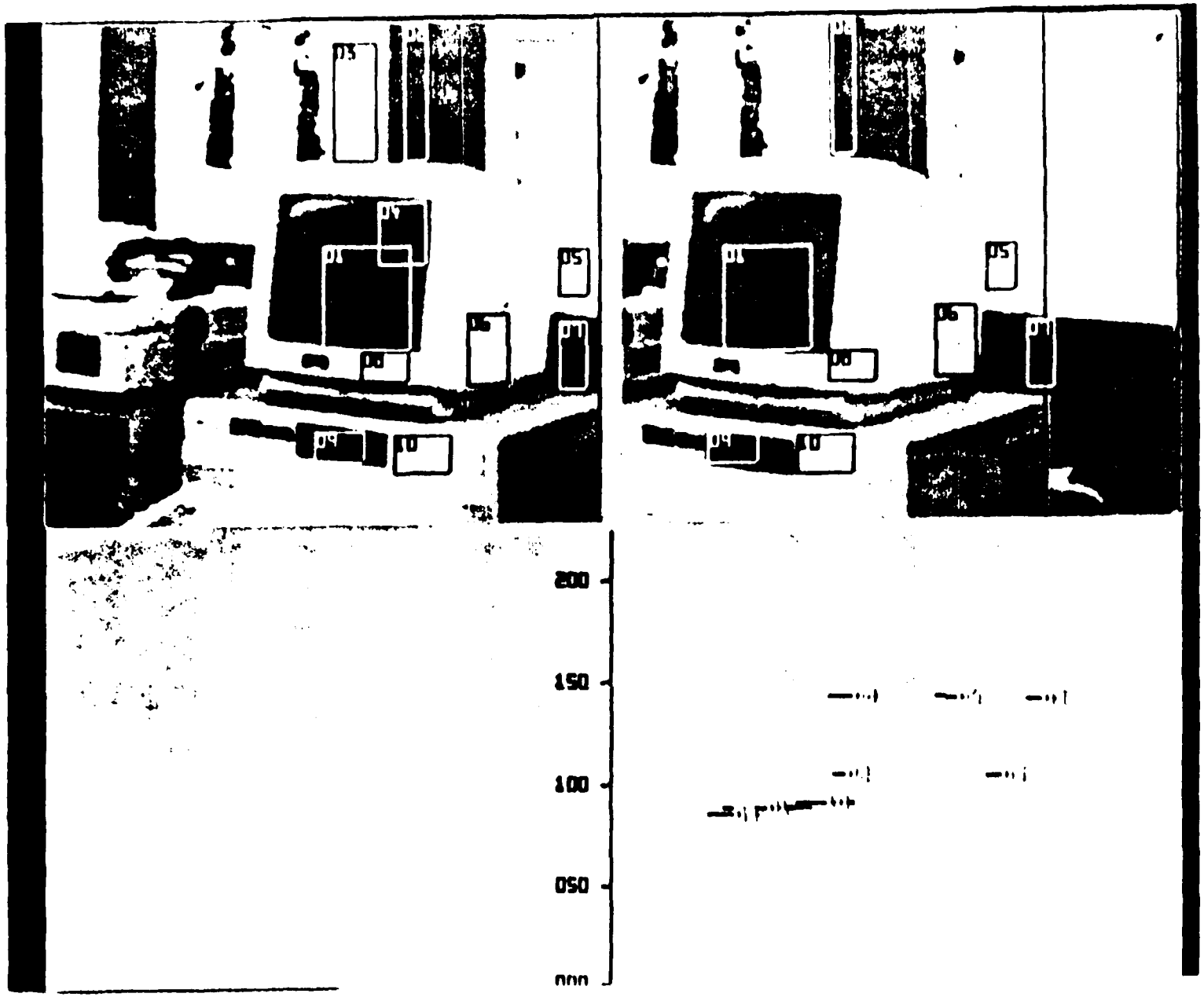


FIGURE A-2 THRESHOLD = 4 AND ITERATION = 2

FIGURE A-3 THRESHOLD = 4 AND QVA ITERATION = 4

FIGURE A-4 THRESHOLD = 4 AND QVA ITERATION = 6

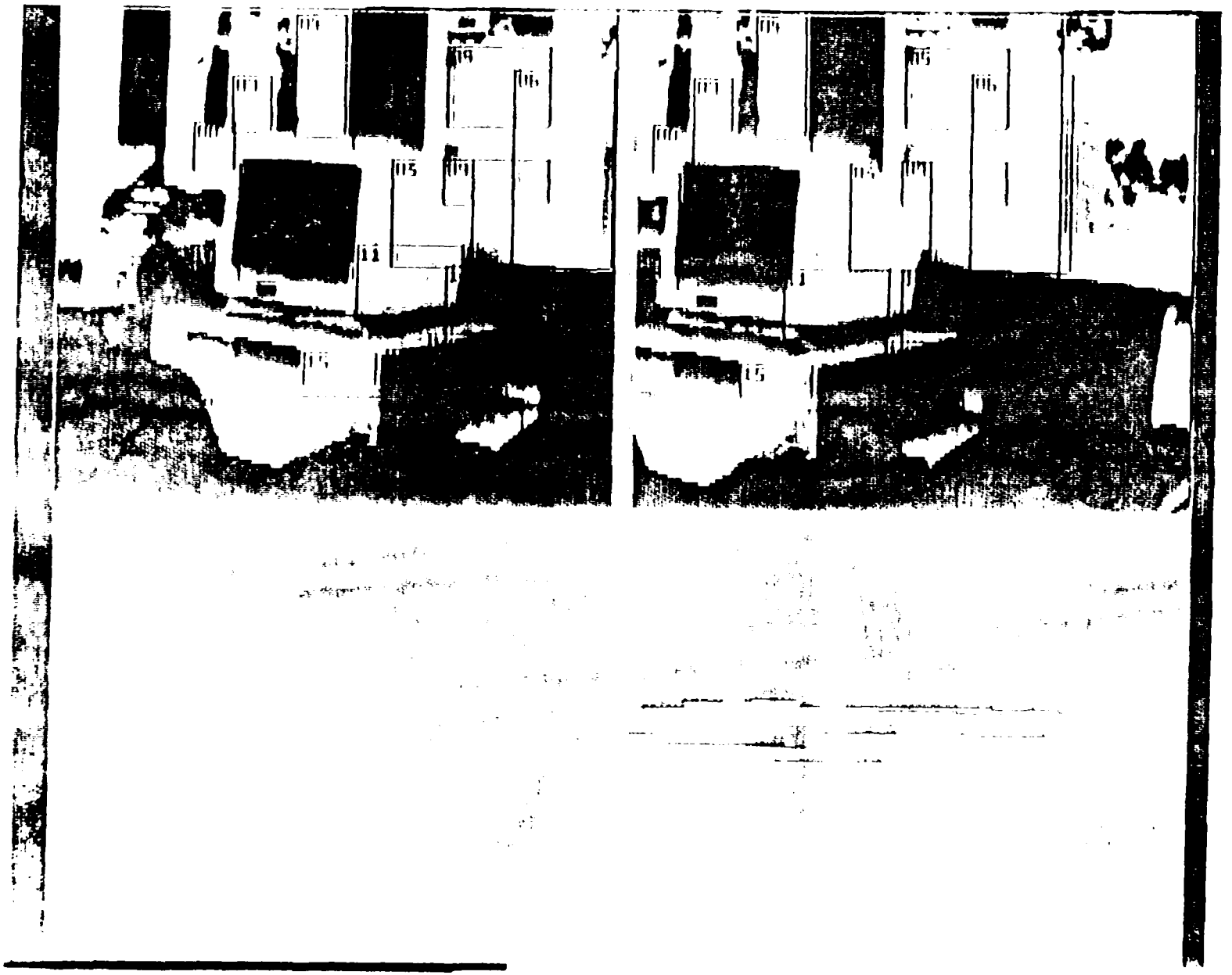


FIGURE A-5 THRESHOLD = 4 AND QVA ITERATION = 8

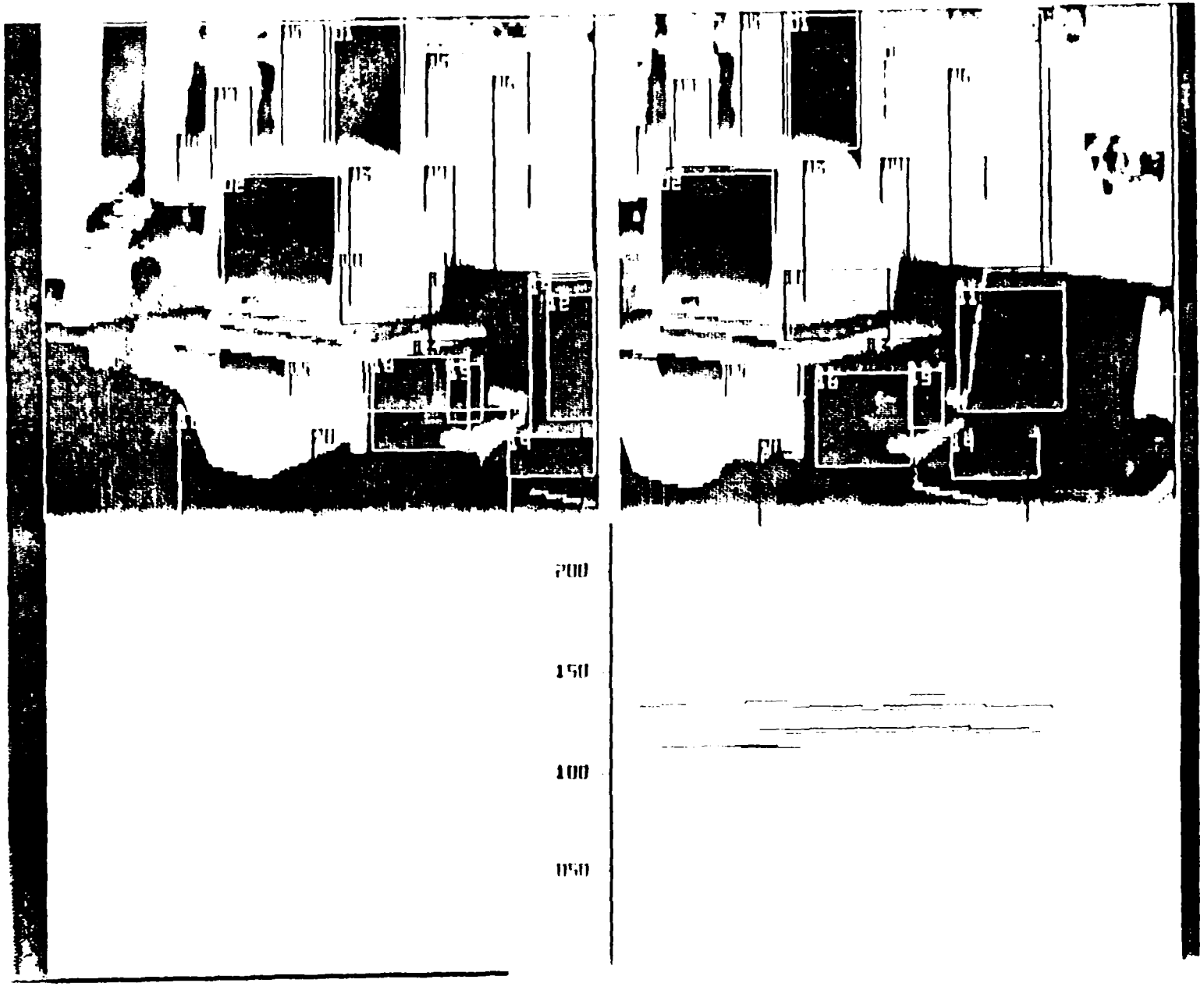


FIGURE A-6 THRESHOLD = 4 AND QVA ITERATION = 10

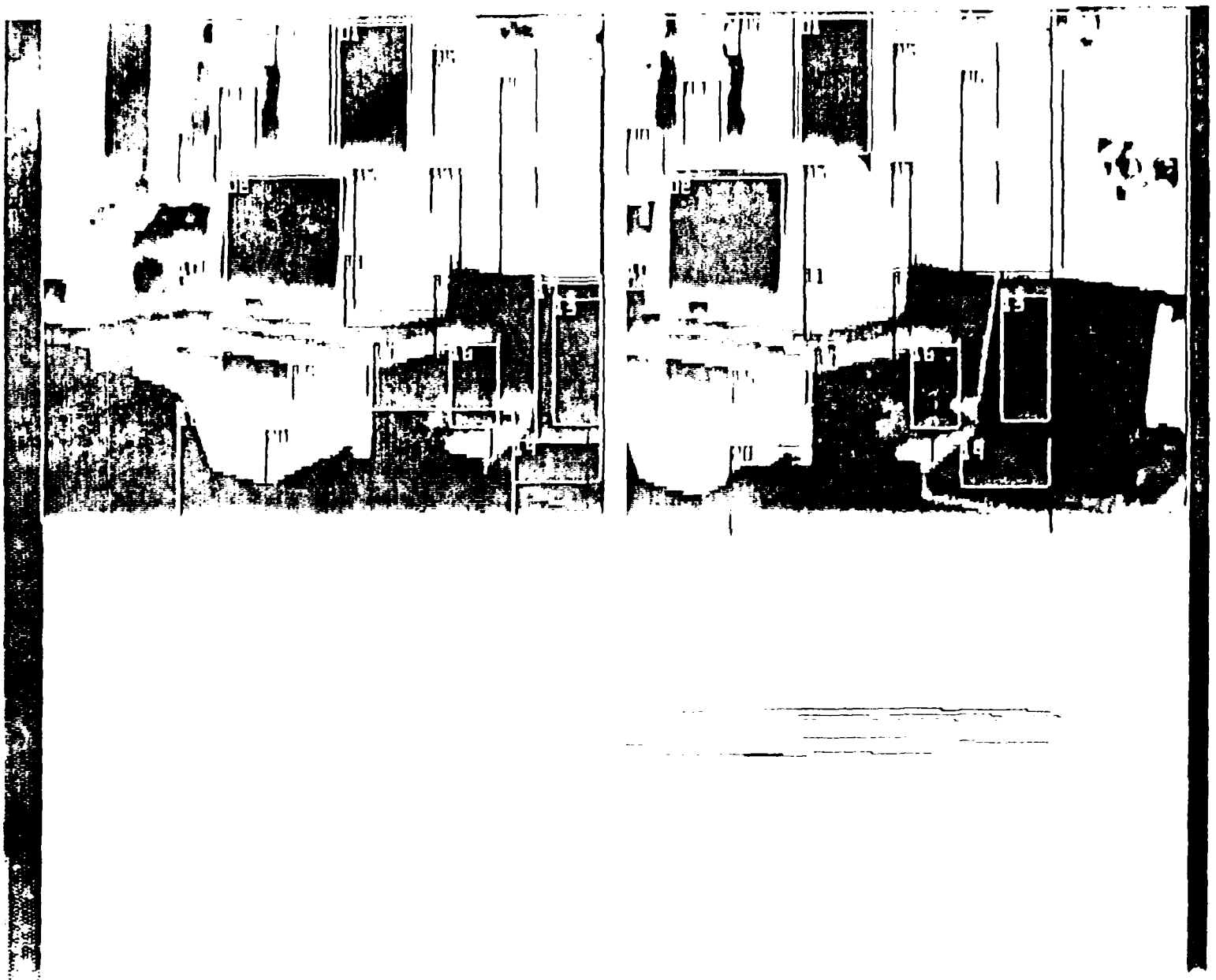


FIGURE A-7 THRESHOLD = 4 AND OVA ITERATION = 12

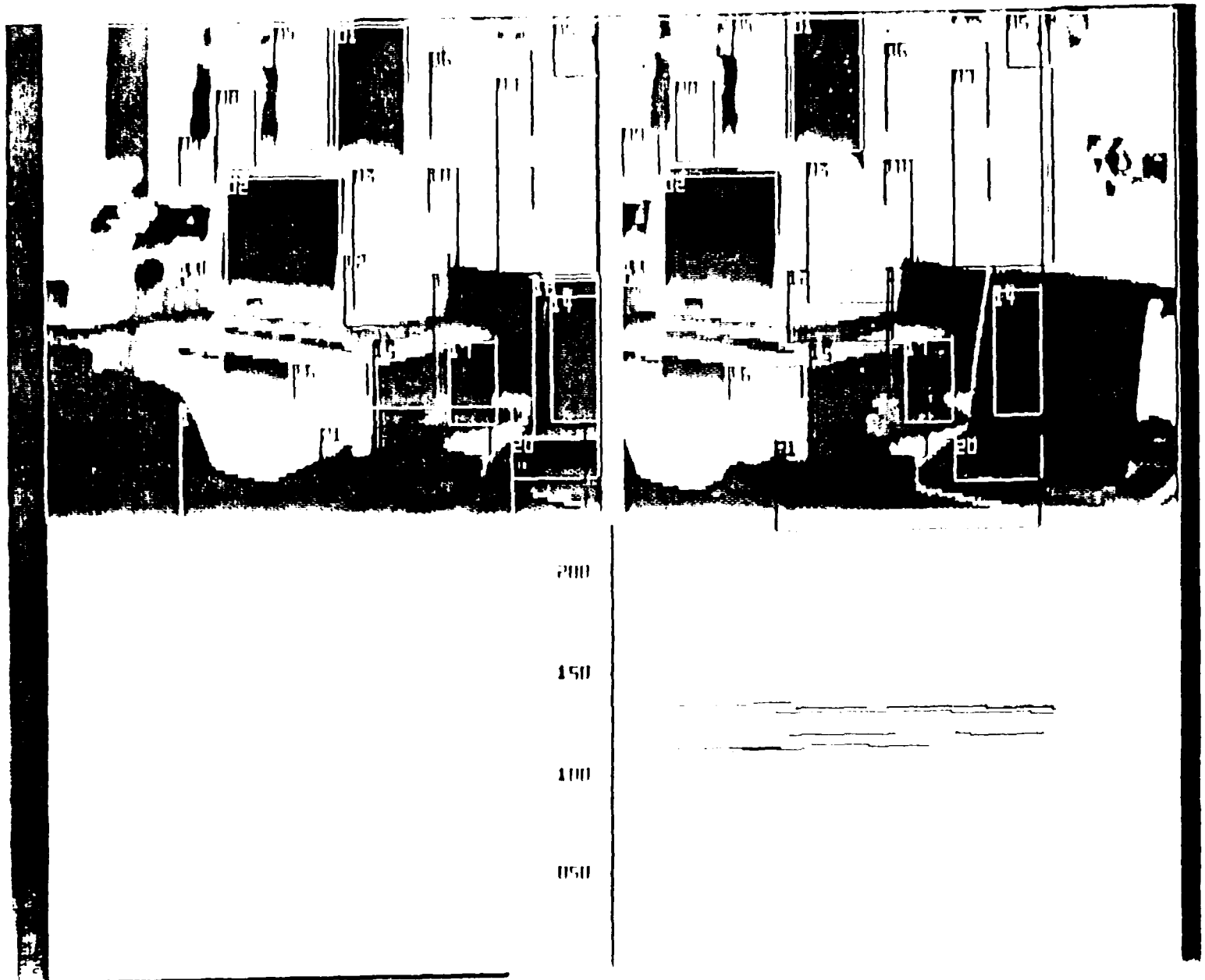


FIGURE A-8 THRESHOLD = 4 AND OVA ITERATION = 14

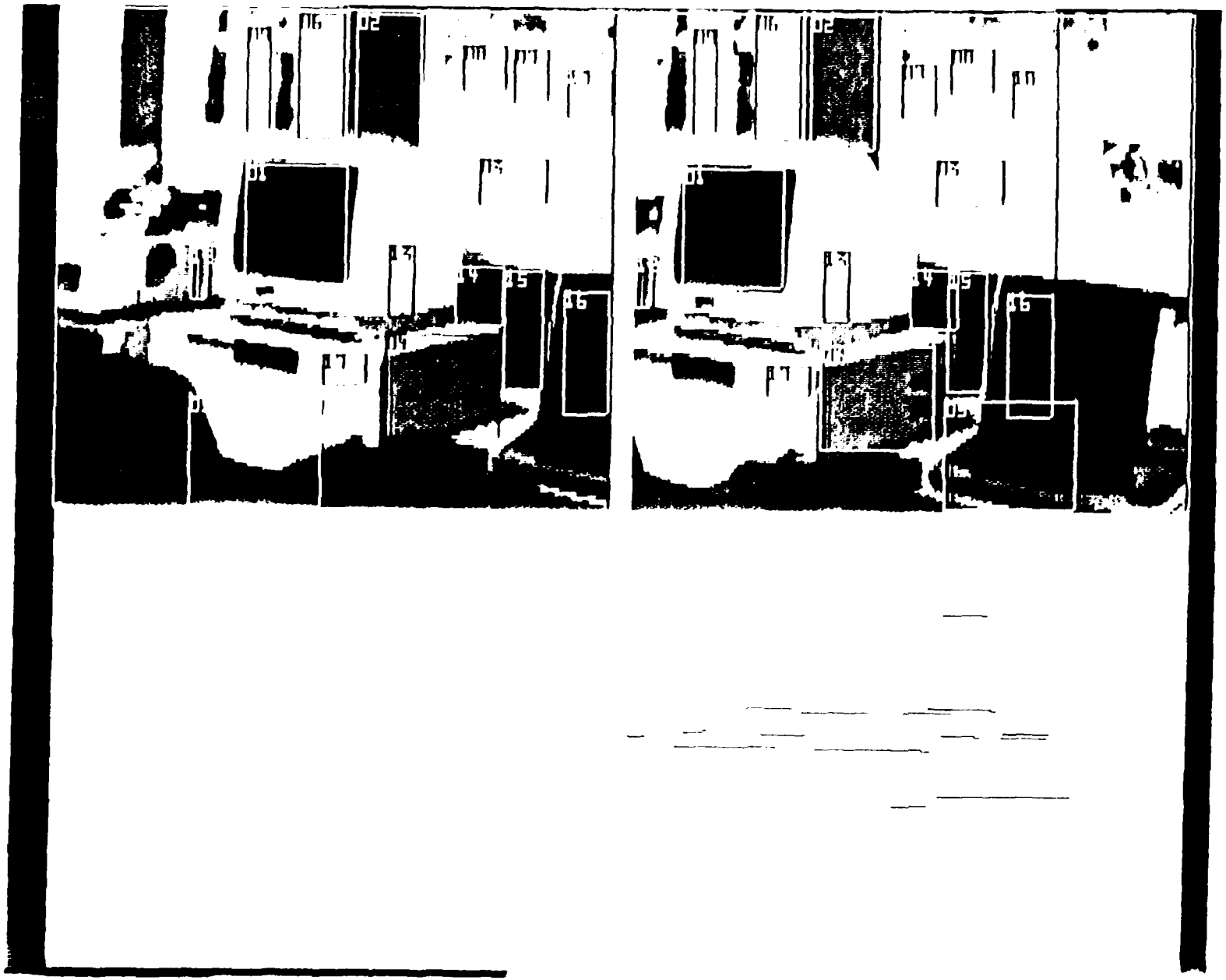


FIGURE A-9 THRESHOLD = 8 AND QVA ITERATION = 2

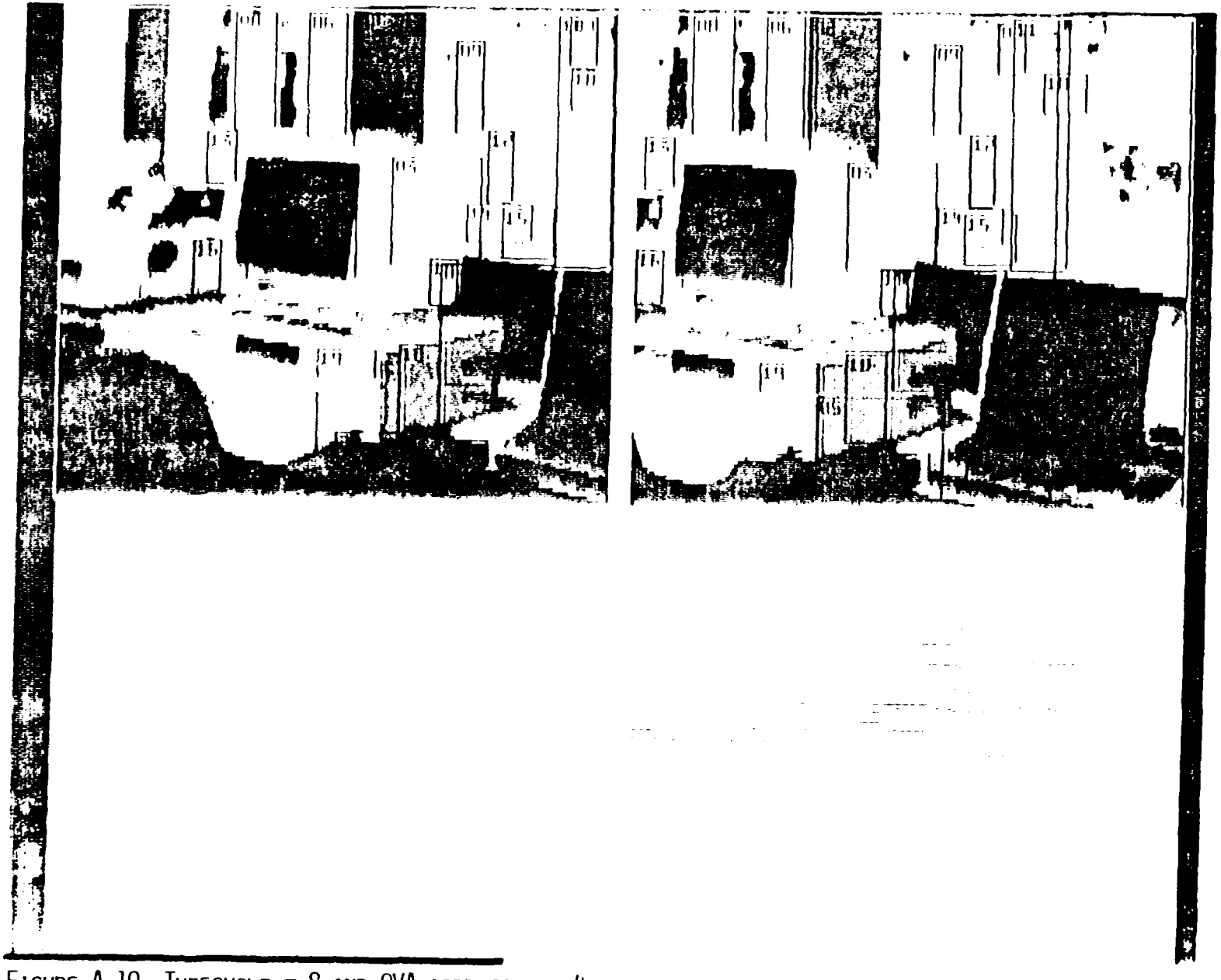


FIGURE A-10 THRESHOLD = 8 AND QVA ITERATION = 4

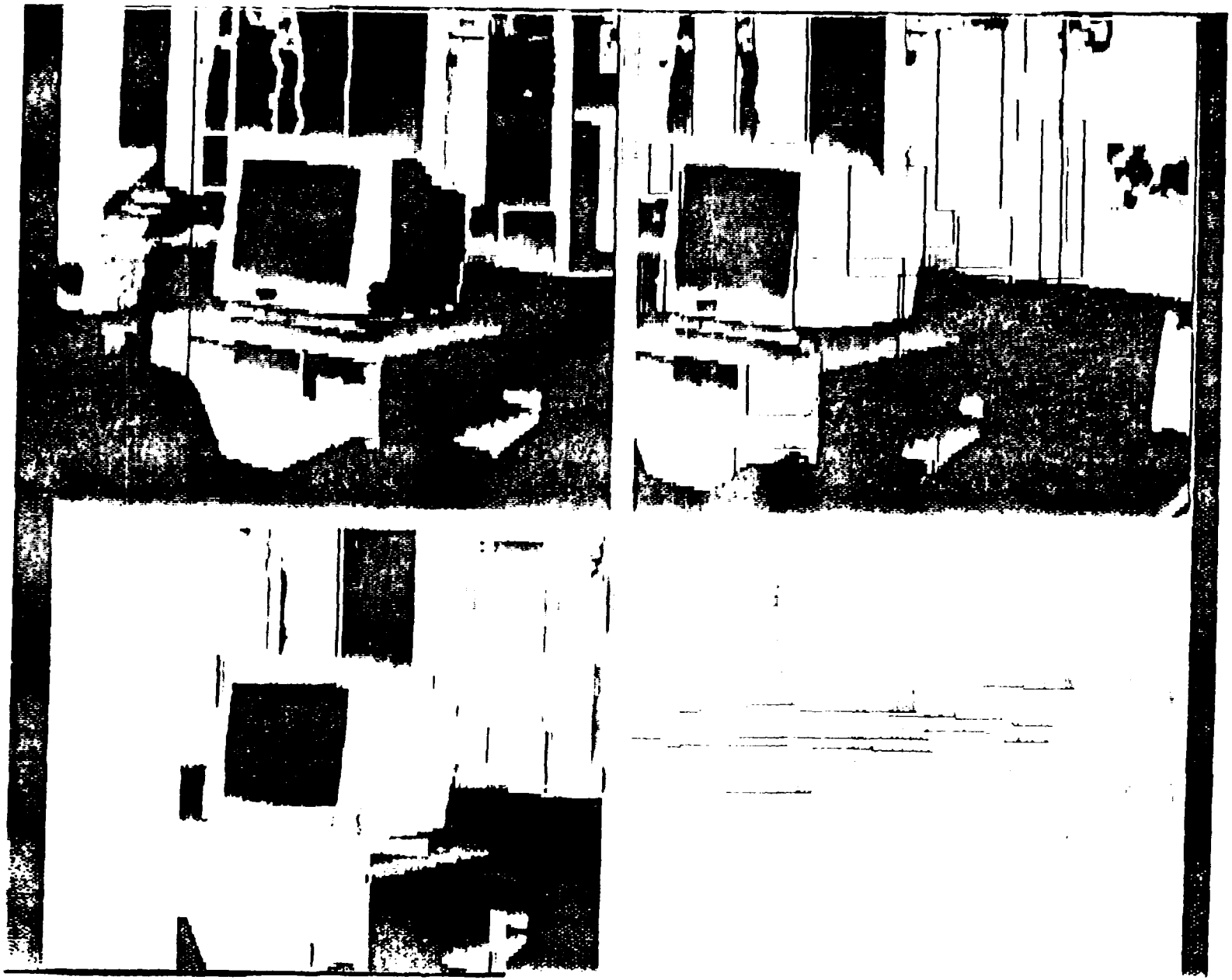


FIGURE A-11 THRESHOLD = 8 AND QVA ITERATION = 6

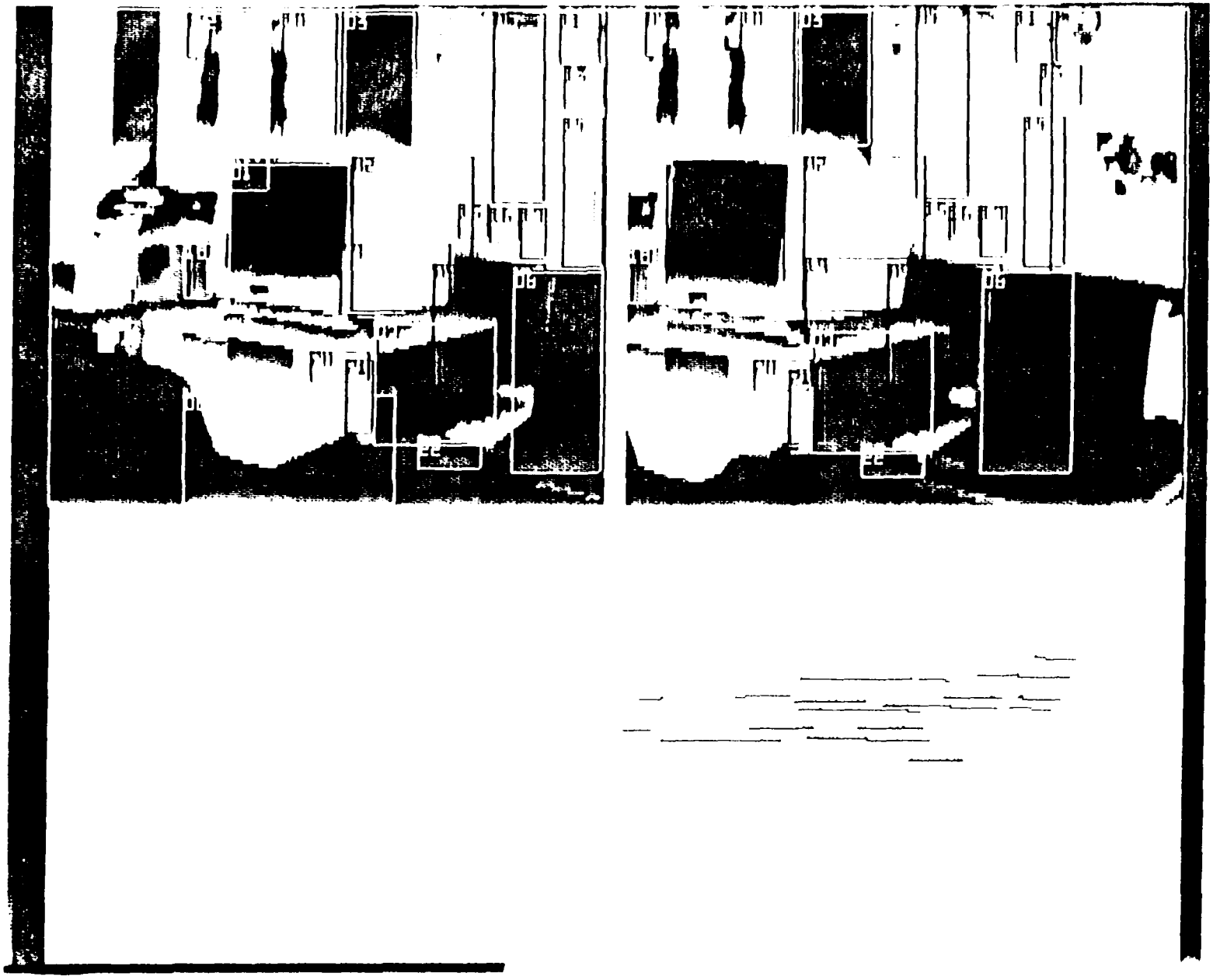


FIGURE A-12 THRESHOLD = 8 AND QVA ITERATION = 8

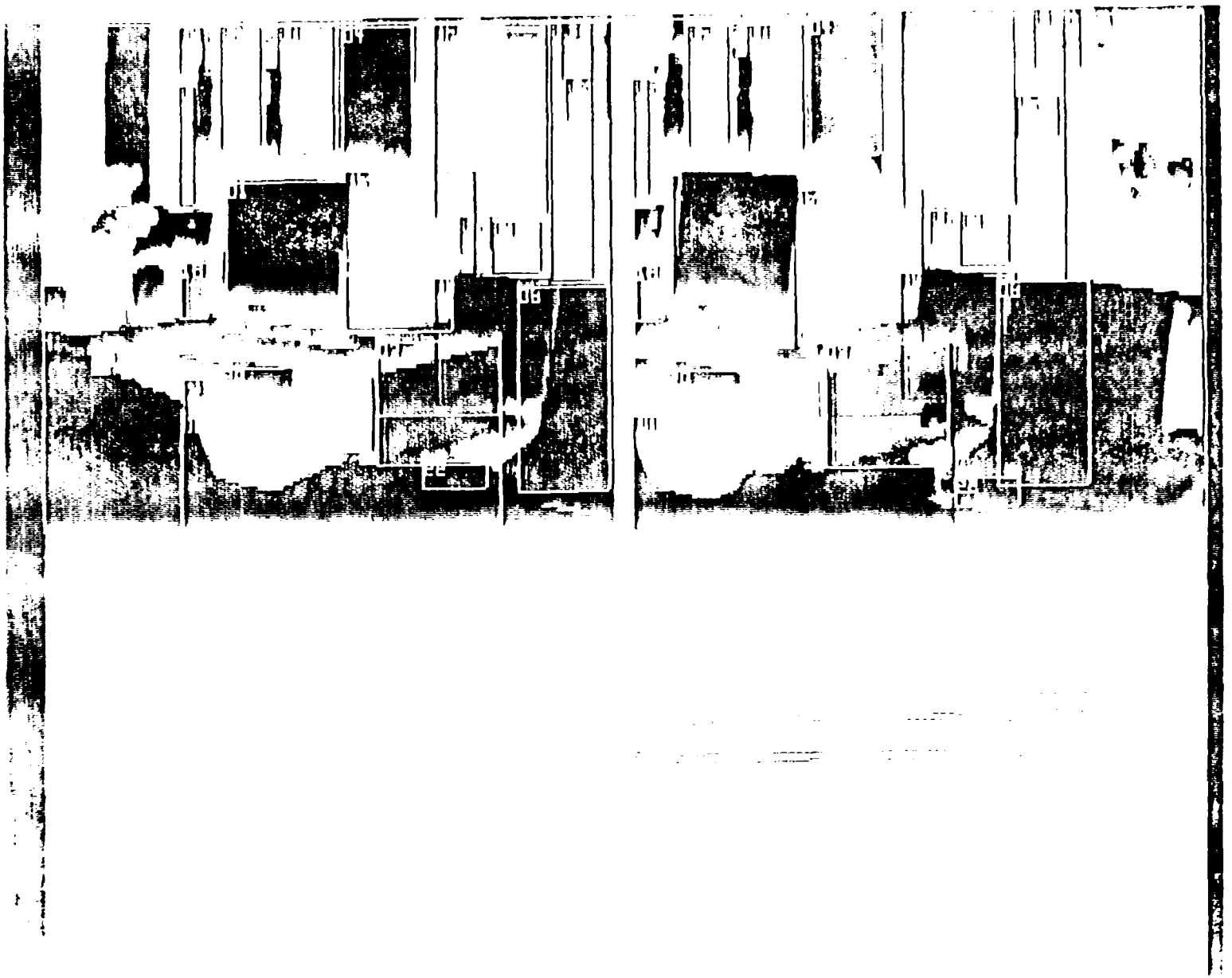


FIGURE A-13 THRESHOLD = 8 AND QVA ITERATION = 10

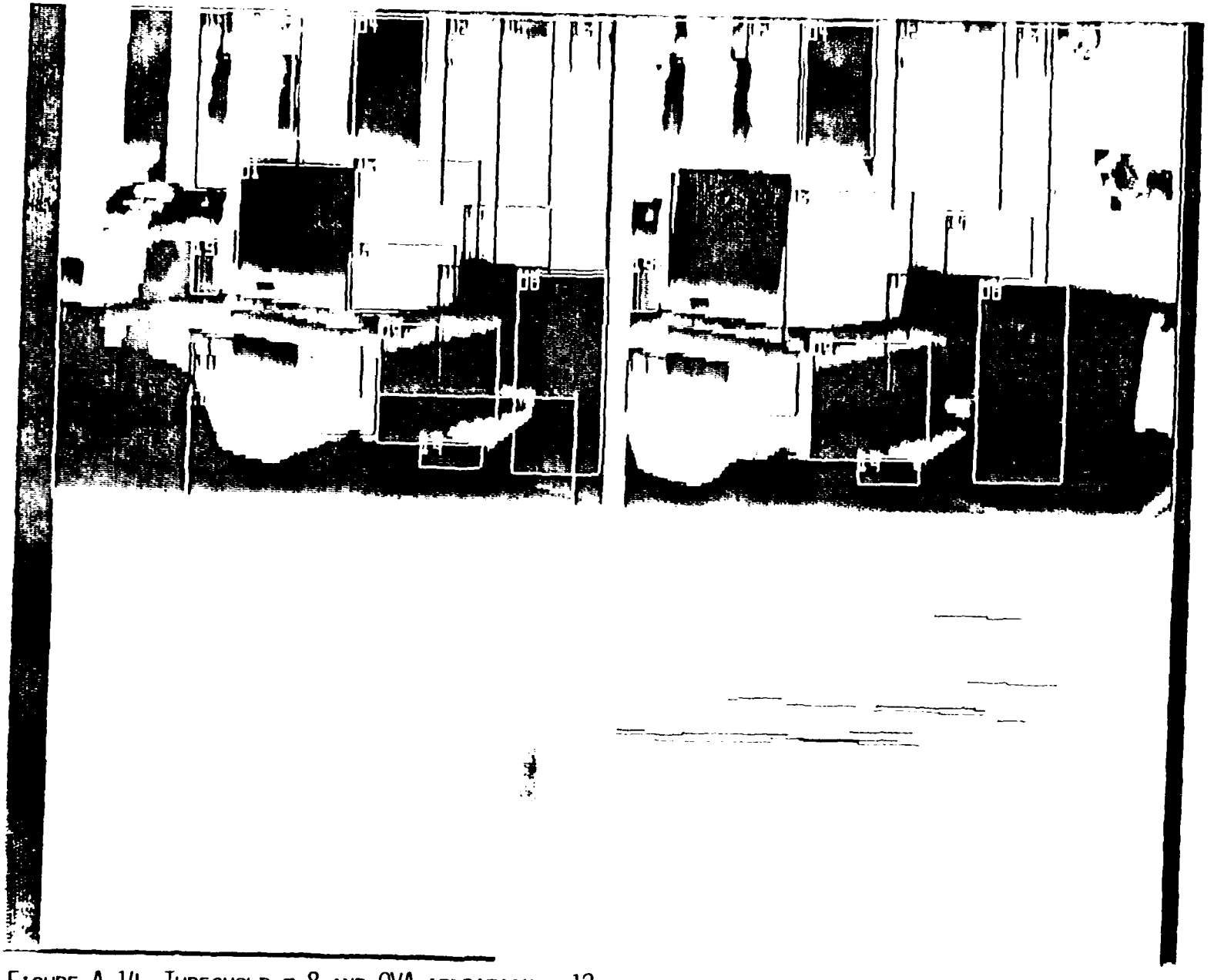


FIGURE A-14 THRESHOLD = 8 AND QVA ITERATION = 12

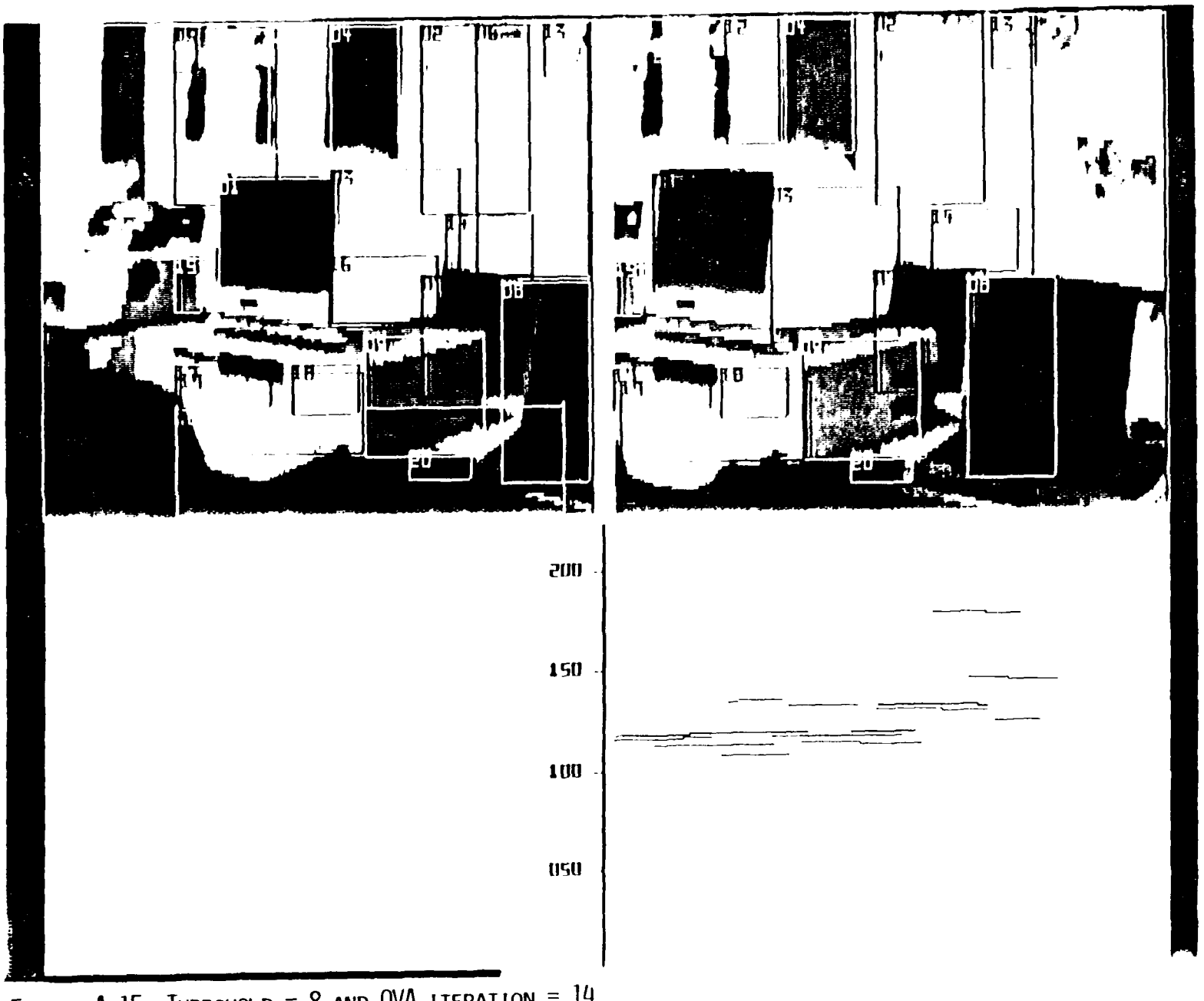


FIGURE A-15 THRESHOLD = 8 AND QVA ITERATION = 14

FIGURE A-16 THRESHOLD = 16 AND QVA ITERATION = 2

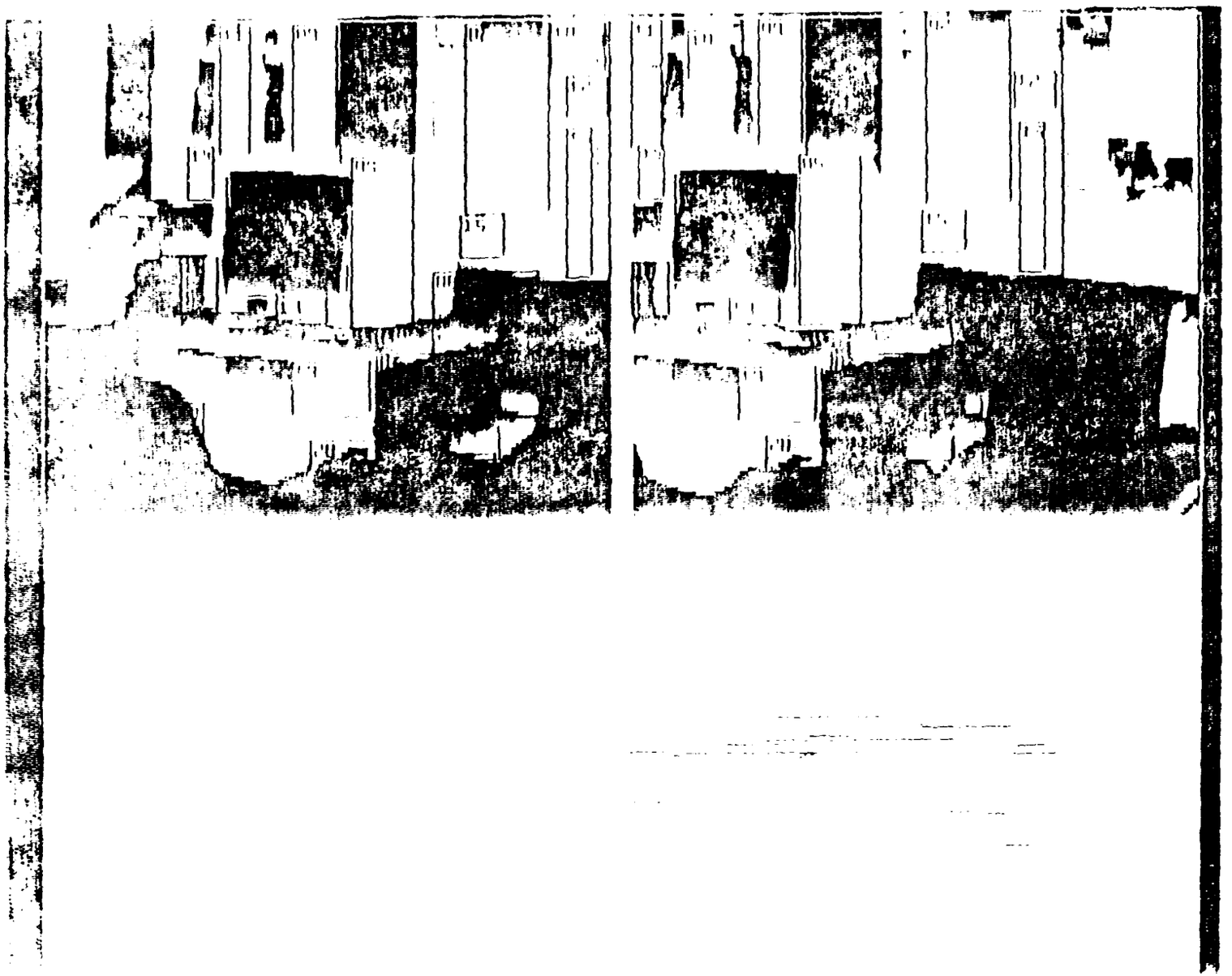


FIGURE A-17 THRESHOLD = 16 AND QVA ITERATION = 4

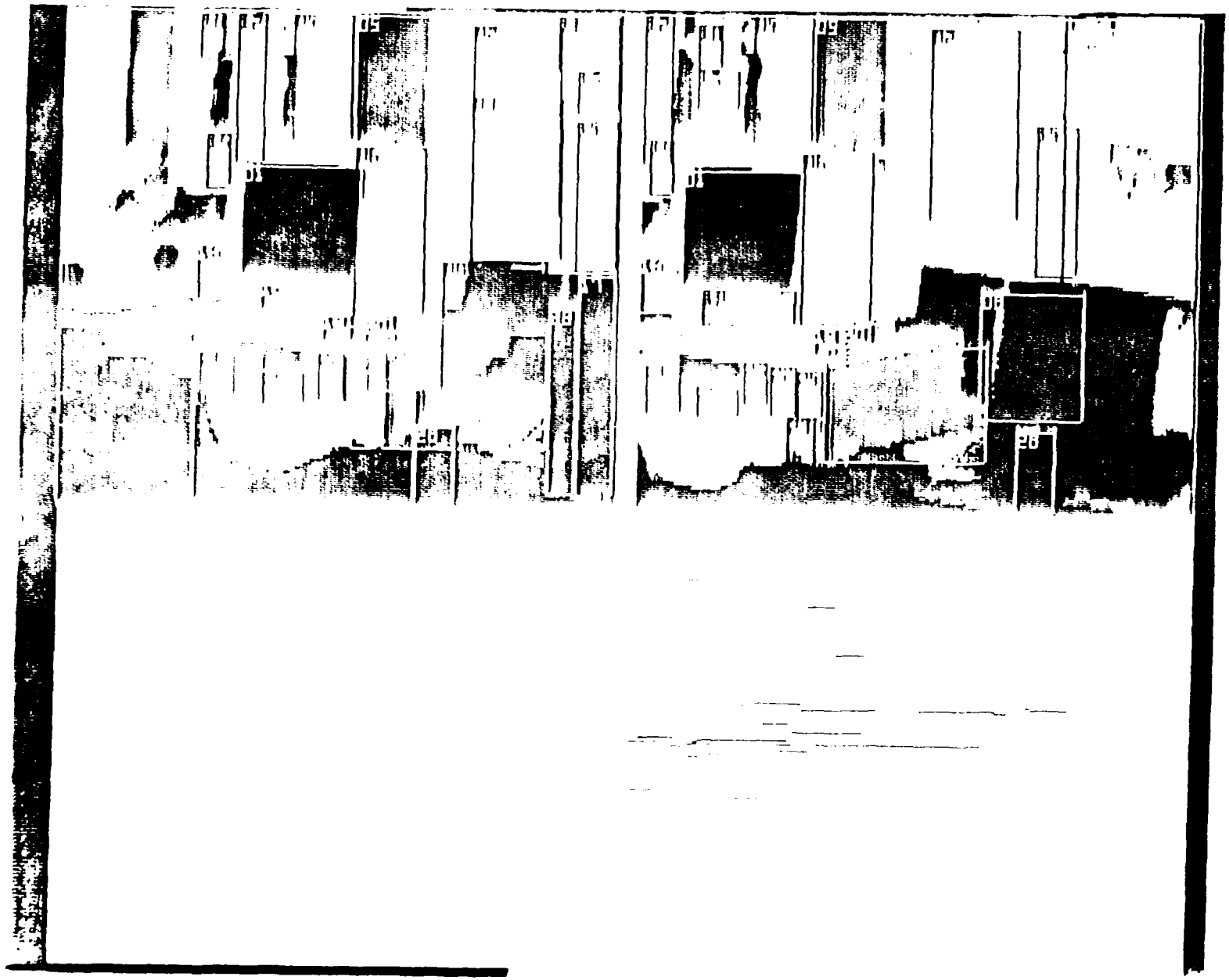


FIGURE A-18 THRESHOLD = 16 AND QVA ITERATION = 6

FIGURE A-19 THRESHOLD = 16 AND QVA ITERATION = 8

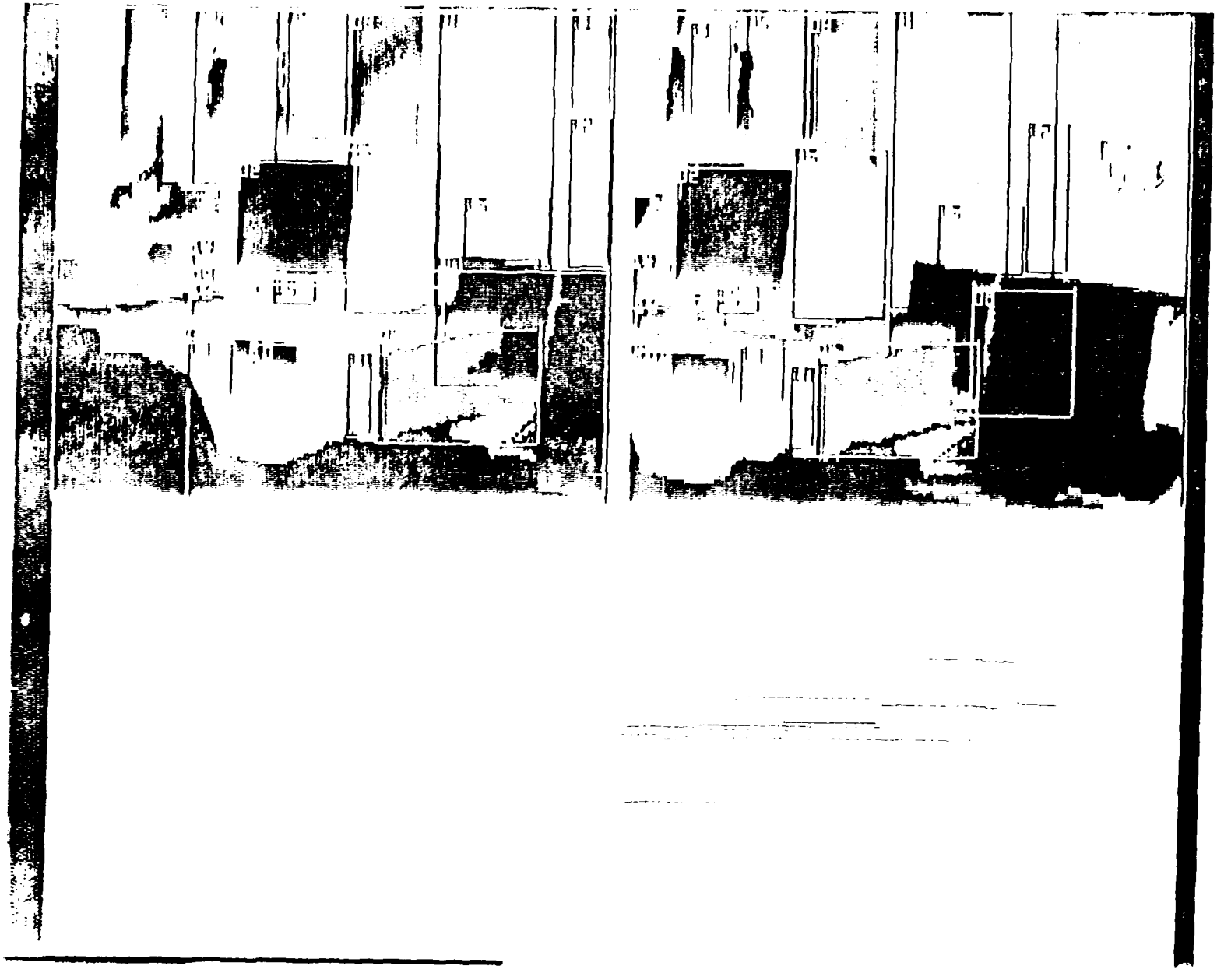


FIGURE A-20 THRESHOLD = 16 AND QVA ITERATION = 10

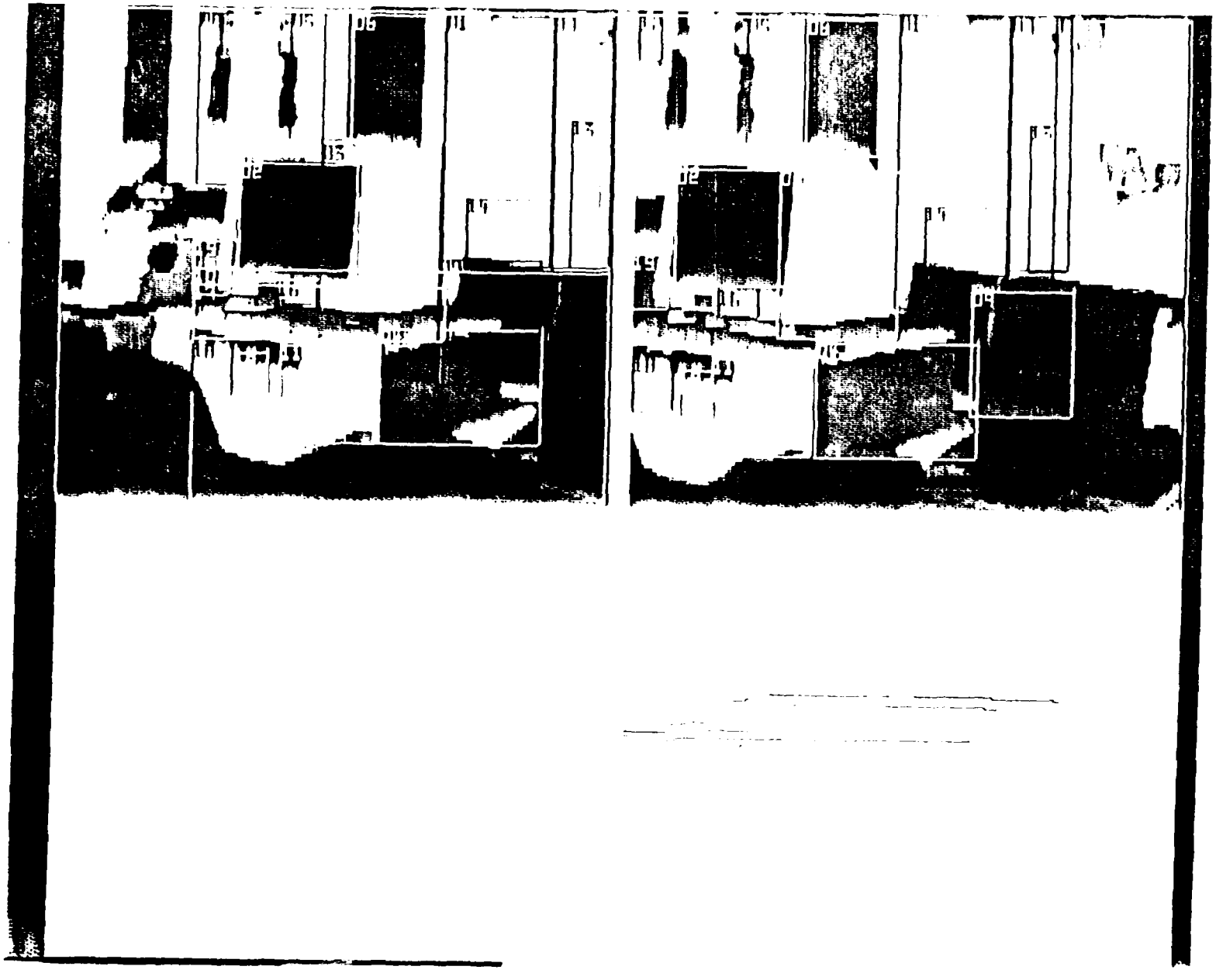


FIGURE A-21 THRESHOLD = 16 AND QVA ITERATION = 12

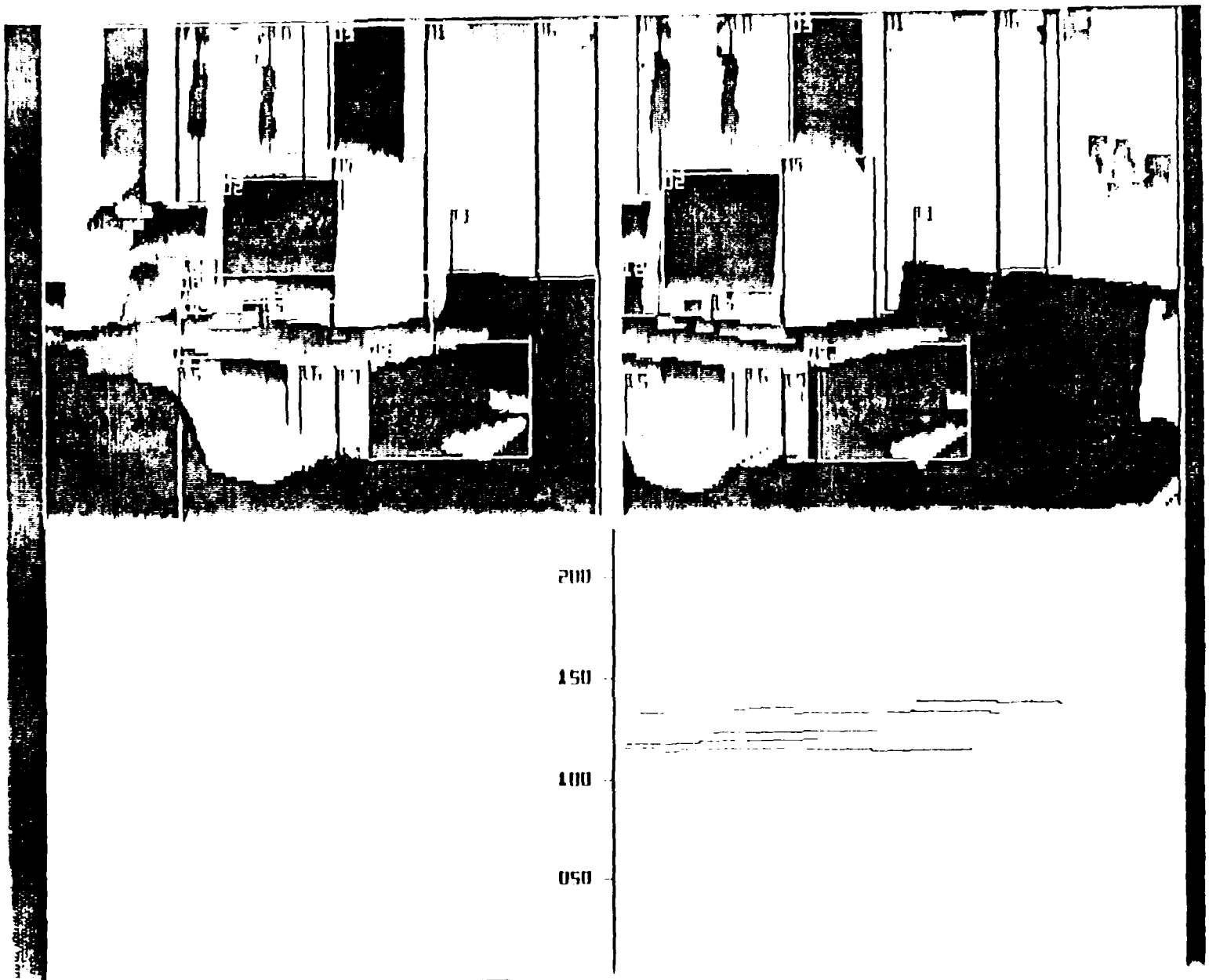


FIGURE A-22 THRESHOLD = 16 AND OVA ITERATIONS = 14

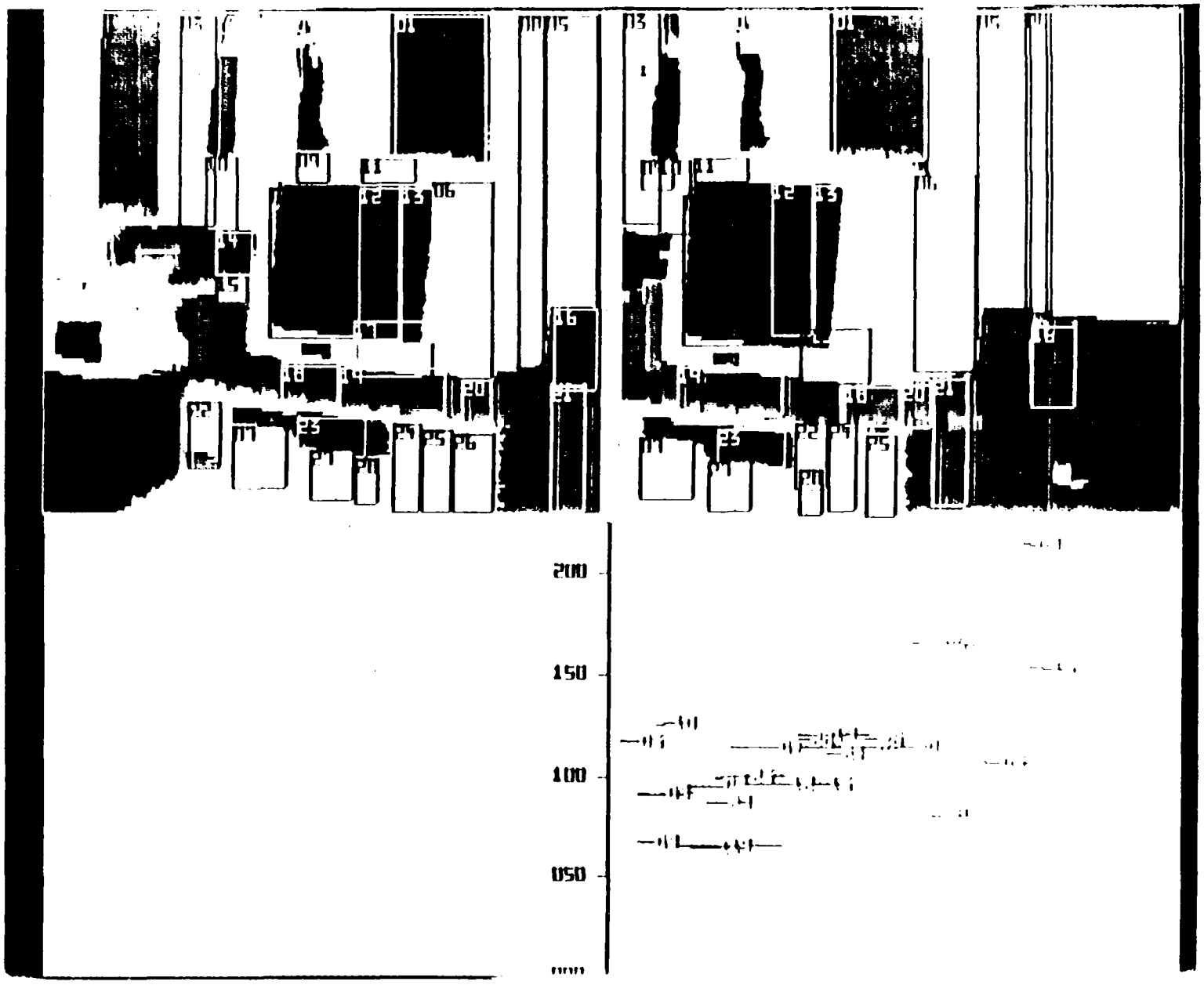
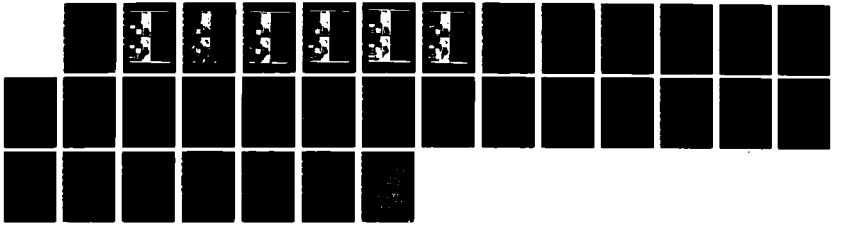
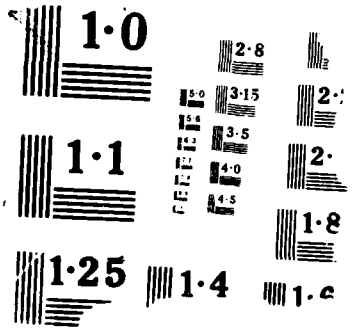


FIGURE A-23 THRESHOLD = 32 AND QVA ITERATION = 4

AD-A193 120 THREE-DIMENSIONAL SCENE ANALYSIS USING STERO BASED IMAGING(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING R E ROBERTS 09 DEC 87 2/2

UNCLASSIFIED AFIT/GE/ENG/87D-54 F/G 17/7 NL




1·0
1·1
1·25
1·4
2·8
3·15
3·5
4·0
4·5
2·
1·8

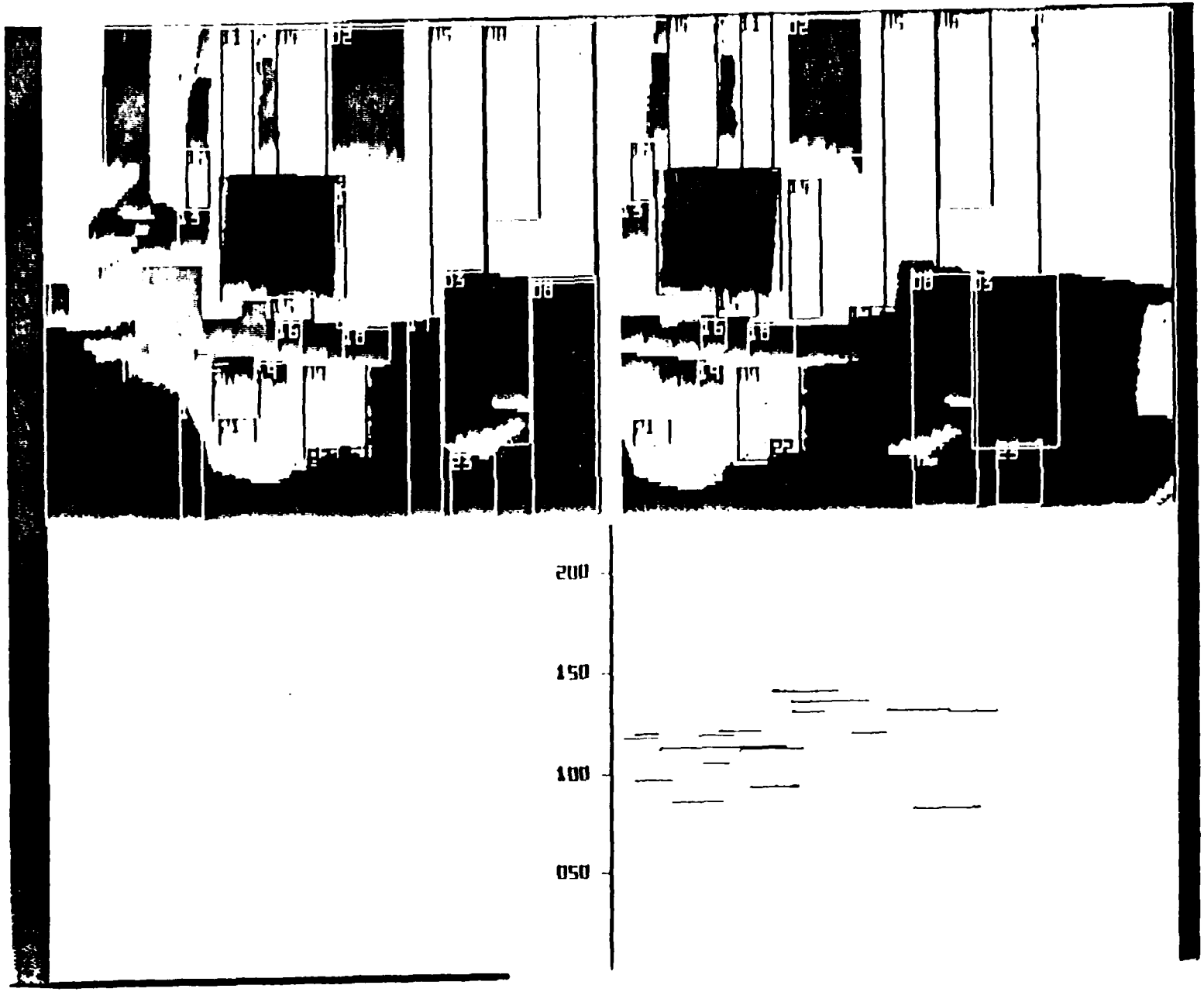


FIGURE A-24 THRESHOLD = 32 AND QVA ITERATION = 4

FIGURE A-25 THRESHOLD = 32 AND QVA ITERATION = 6

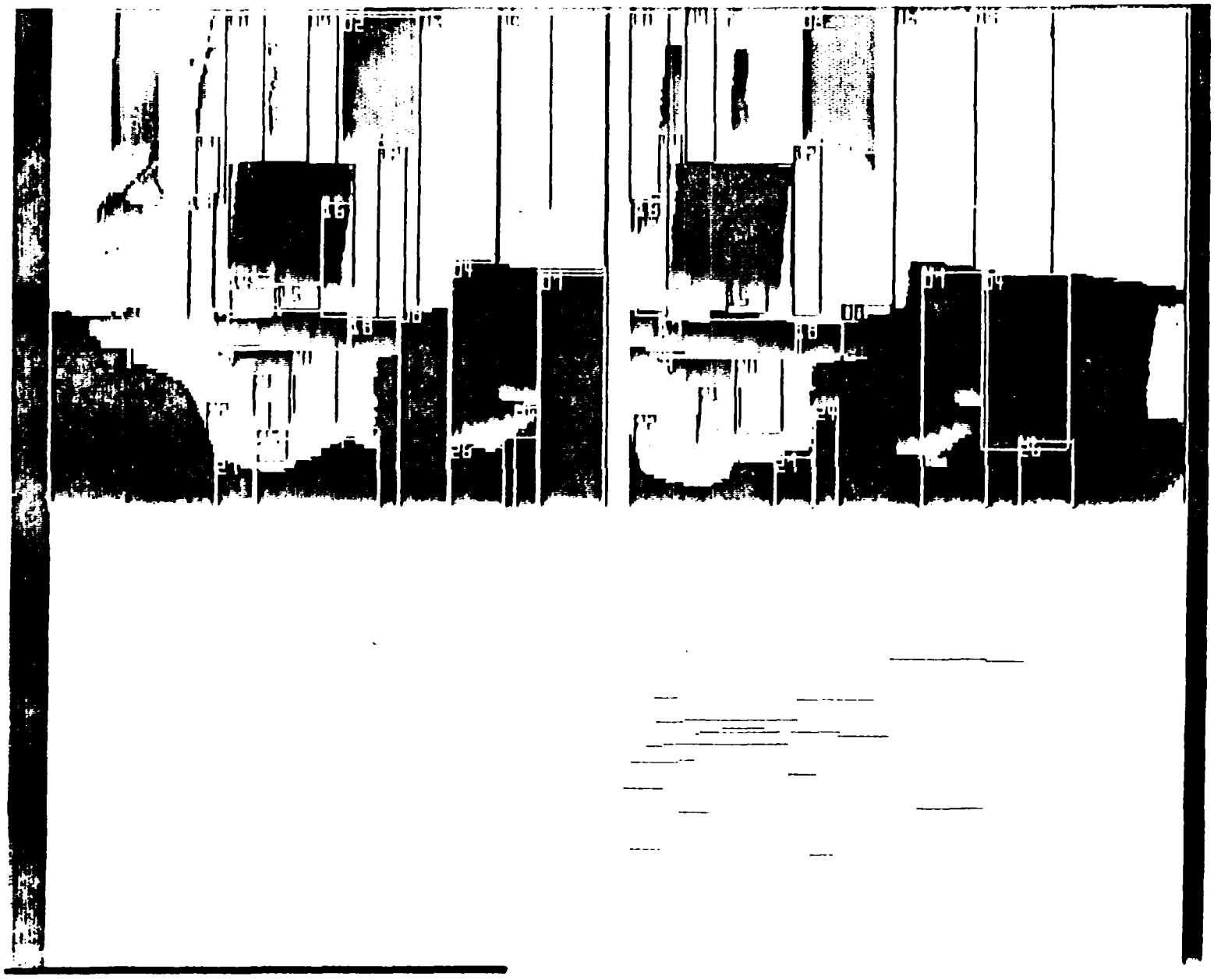


FIGURE A-26 THRESHOLD = 32 AND ITERATION = 8

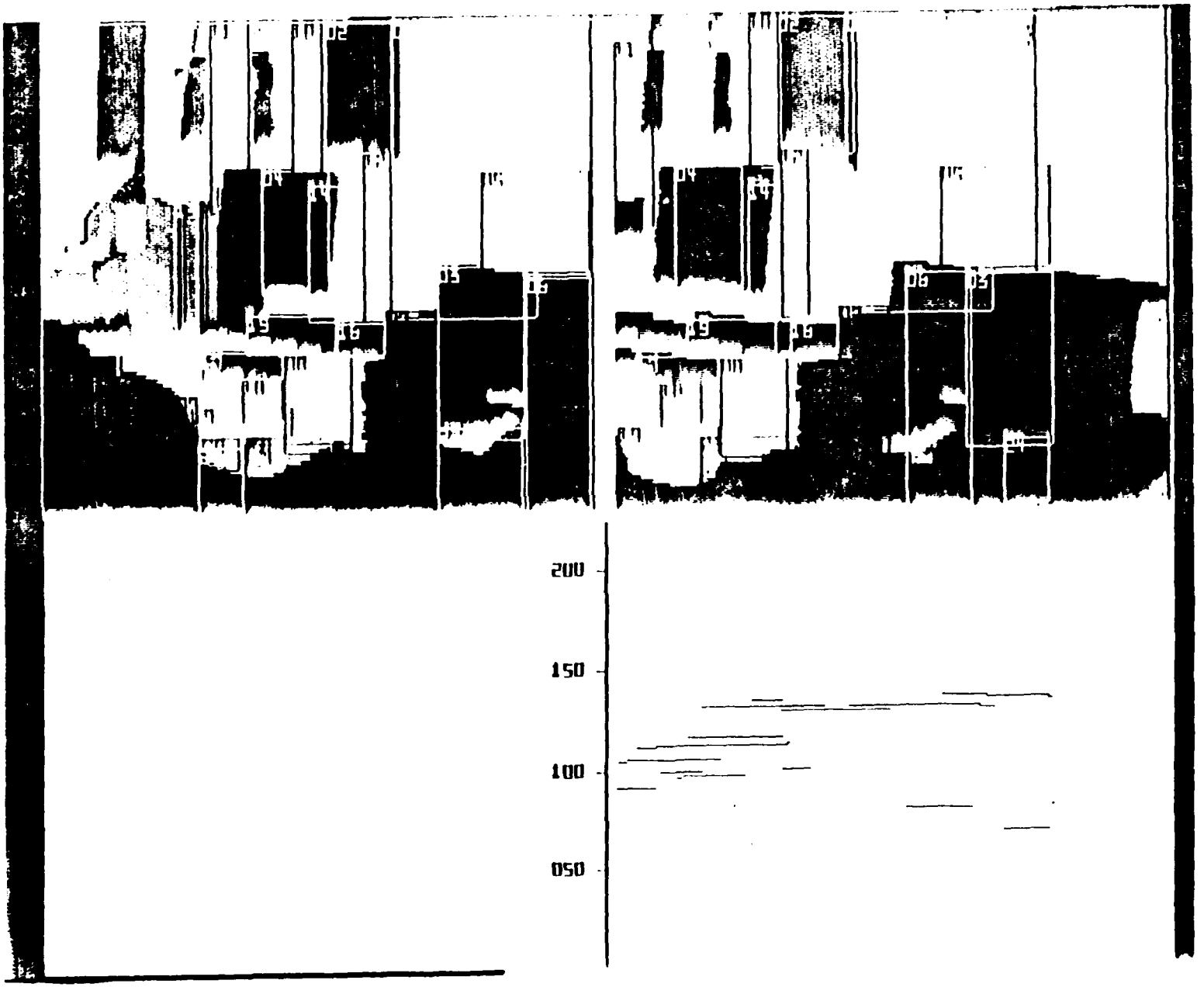


FIGURE A-27 THRESHOLD = 32 AND QVA ITERATION = 10

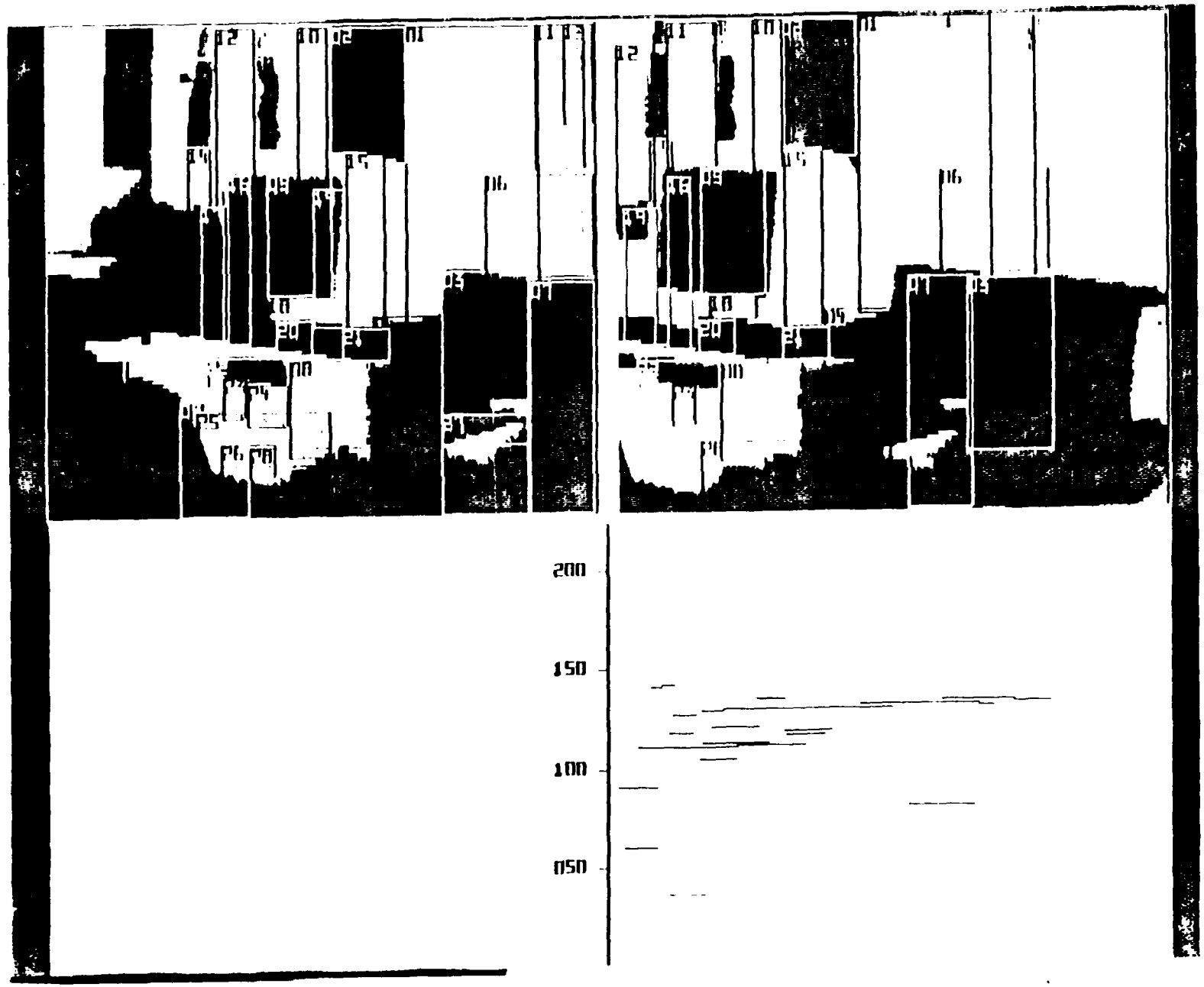


FIGURE A-28 THRESHOLD = 32 AND QVA ITERATION = 12

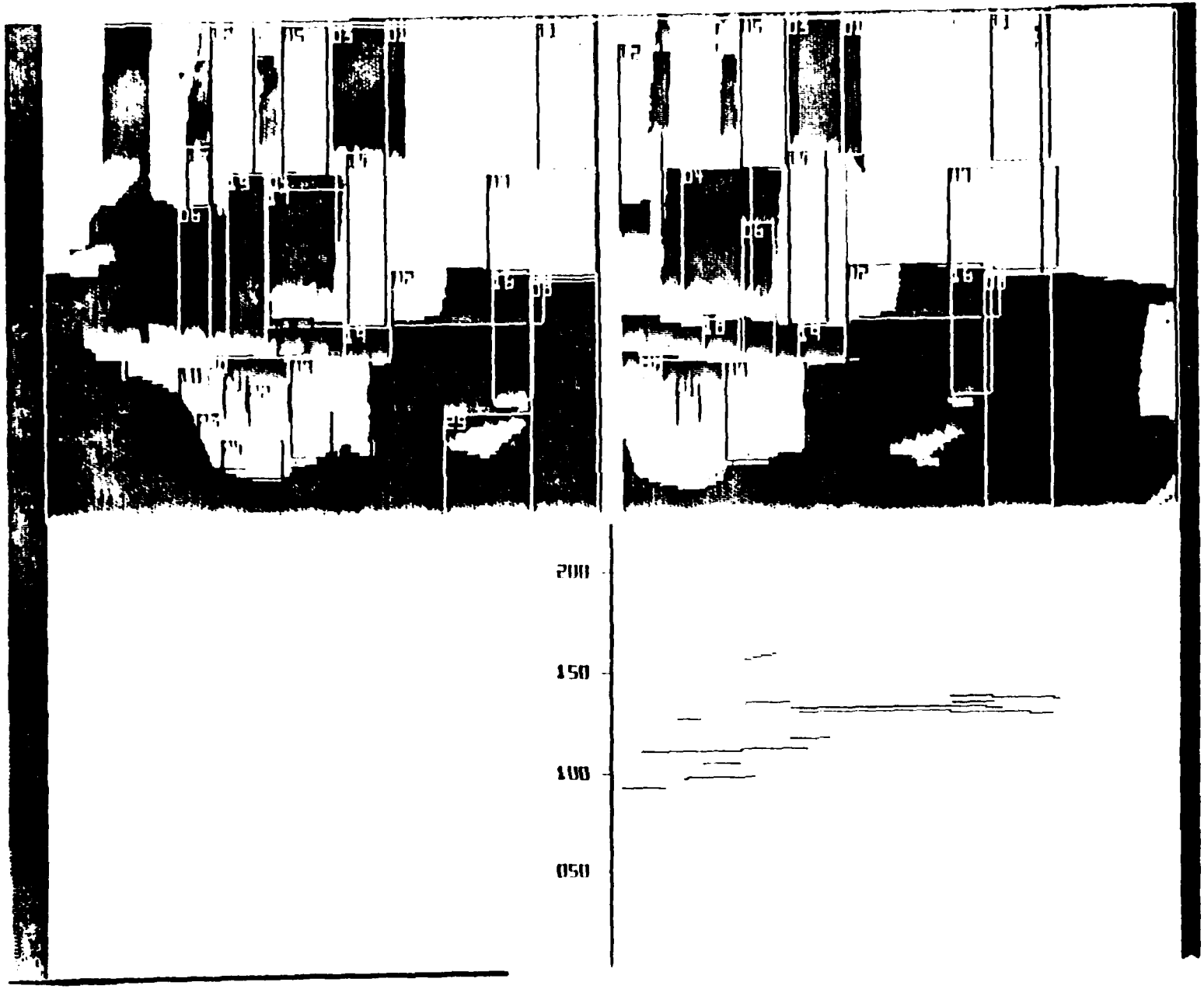


FIGURE A-29 THRESHOLD = 32 AND QVA ITERATION = 14

APPENDIX B

Program Listing for Code Used in Implementing Stereo Vision

1. All programs were written in C programming language. These programs were executed using VAX C version 2.0 on a digital MicroVAX II running VMS operating language.

2. All ITEX100 programs are written in C. These programs are part of Imaging Technology Series 100 family of single-board image processors.

3. A brief outline of the programs used in this thesis is included below. Page numbers are included for quick reference.

```
/*************************************************************************/
/*                                                                       */
/*      Program Name    IMAGE_TEST.C                                     */
/*                                                                       */
/*      Author : Richard E. Roberts     Jun 1987                         */
/*                Special thanks to Larry Lambert for assisstance        */
/*                                                                       */
/*      Test program for edging algorithms.                              */
/*      This program provides a menu for the users that will enable      */
/*      them to perform the following tasks:                             */
/*          1. Initialize the ITEX100 hardware and monitor               */
/*          2. Aim and ficus both left and right cameras                 */
/*          3. Acquire two images  -  left and right                     */
/*          4. Save images on the monitor to disk memory                 */
/*          5. Recal images from disk memory and write to the monitor    */
/*                                                                       */
/*************************************************************************/

#include "sys$library:stdio.h"
#include "dua0:[iti100.itex]stdtyp.h"
#include "dua0:[iti100.itex]itex100.h"

static int      option;

main()           /****   START OF MAIN PROGRAM   ****/
{

int   x, y, dx, dy, color;
int   comment[255], name[256];
char  test[10];

/*  initialize ITEX100 board and monitor  */
unsigned        base = 0x1600;
long            mem = 0x200000L;
int             flag = 1, block = 8;
int             errnum;
sethdw(base, mem, flag, block);
/*************************************************************************/

while (1)
        {
        cls();
        printf("\n         ****  EDGE ALGORITHM TEST PROGRAM ****\n");
        printf("\n              0:Quit to VMS\n");
        printf("\n              2:Initialize hardware\n");
        printf("\n              3:Aim and focus left and right cameras");
        printf("\n              4:Acquire left and right images");
        printf("\n              5:Save images to disk");
        printf("\n              6:Recall images from disk\n");
        printf("\n\n\n\n\n Select an option by its number\n>");
        scanf("%d",&option);

        switch (option) {
```

```
case 0:
        setcamera(0);
        return;

case 2:
        initialize();
        sclear(100);
        break;

case 3:
        setcamera(2);              /* select left camera image */
        grab(0);
        cls();
        printf("\nPress RETURN when satisfied with left image.");
        getchar();
        getchar();
        setcamera(1);             /* select right camera image */
        grab(0);
        printf("\nPress RETURN when satisfied with right image.");
        getchar();
        break;

case 4:
        cls();
        printf("\nPress RETURN to initiate image acquisition.");
        getchar();
        getchar();

        setcamera(2);              /* select left camera image */
        printf("\n Camera in use = %d",getcamera());
        grab(0);
        printf("\nPress RETURN to take left camera picture.\n");
        getchar();
        stopgrab(1);
        cls();
        printf("saving temporary left image....");
        saveim(131,131,250,250,0,"temp_left.pic","no com");

        setcamera(1);             /* select right camera image */
        printf("\n Camera in use = %d",getcamera());
        grab(0);
        printf("\nPress RETURN to take right camera picture.\n");
        getchar();
        stopgrab(1);
        printf("\n\nsaving temporary right image....");
        saveim(131,136,250,250,0,"temp_right.pic","no com");

        sclear(100);             /* put both images on monitor */
        readim(1,1,250,250,"temp_left.pic","no com");
        readim(260,1,250,250,"temp_right.pic","no com");

        delete("temp_left.pic;1");
        delete("temp_right.pic;1");
        break;
```

```
        case 5:
                cls();
                printf("\n\nEnter image filename (including EXT)\n>");
                scanf("%s",name);
                printf("\nSaving both images.  under single filename");
                errnum = saveim(0,0,512,252,0,name,"none");
                if (errnum < 0)
                {
                  printf("\nUnable to open file or write to the disk.");
                  printf("\nPress RETURN to continue.");
                  getchar();
                  getchar();
                }
                waitvb();
                break;

        case 6:
                cls();
                printf("\n\nEnter image filename (including EXT)\n>");
                scanf("%s",name);
                errnum = readim(0,0,512,252,name,"none");
                if (errnum < 0)
                {
                  printf("\nFile not found. Press RETURN to continue.");
                  getchar();
                  getchar();
                }
                break;

        default:
                cls();
                printf("\n You did not select a valid option number.");
                printf("\n\n Press RETURN to continue.");
                getchar();
                getchar();
                break;

  }     }

}                   /*****   END OF MAIN PROGRAM    ****/


/*************************************************************************/
/*  Subroutine cls used as clear screen during interactive opertaion  */
/*************************************************************************/
cls()
{
   printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
   return;
}


/************************   END OF IMAGE_TEST.C   ************************/
```

```
/*************************************************************************
*                                                                        *
*       Program Name :  LR_QVA.C                                         *
*       Author       :  Richard E. Roberts      Jul 1987                 *
*                                                                        *
*       Special thanks to Larry Lambert for his invaluable               *
*       assistance during the writting and debugging of this code.       *
*                                                                        *
*************************************************************************/

#include "sys$library:stdio.h"
#include "dua0:[iti100.itex]stdtyp.h"
#include "dua0:[iti100.itex]itex100.h"

/*************************************************************************/
/*            GLOBAL VARIABLES USED IN THE PROGRAM                       */
/*************************************************************************/
static int      option;       /* used menu option selection variable  */
int             thrsh;        /* noise threshold variable             */



/*************************************************************************/
/*                    START MAIN PROGRAM                                 */
/*  Raw pixel data is input from the monitor for processing              */
/*  Left image in coordinates x=0 y=0 thru x=250 y=250                   */
/*  Right image in coordinates x=260 y=0 thru x=510 y=250                */
/*************************************************************************/
main()
{

int     lpic[250][250];     /* origional left image pixel data          */
int     rpic[250][250];     /* origional right image pixel data         */
int     name[256];          /* filename entered to store QVA results    */

           /*   Initialize ITEX100 board and monitor   */
unsigned        base = 0x1600;
long            mem = 0x200000L;
int             flag = 1, block = 8;
int             errnum;
int             i, j, l, m, n;
sethdw(base, mem, flag, block);

cls();
printf("\n       ******  LEFT AND RIGHT QVA RESULTS    ******");
printf("\n\n\n\n\n\n\nPlease wait while left and right images");
printf(" are separated and stored.... ");
for(j=0; j<250; j++)          /* obtain left and right images from   */
{                             /* video RAM memory                    */
   for(i=0; i<250; i++)
   {
      l=i+1;
      m=i+261;
      n=j+1;
```

```
      lpic[i][j] = rpixel(l,n);      /* left image pixel color array  */
      rpic[i][j] = rpixel(m,n);      /* right image pixel color array */
    }
  }
  qva_menu(lpic,rpic);       /* enter test loop for matching algorithms */

  }                 /****  END OF MAIN PROGRAM ****/



  /*************************************************************************/
  /* Subroutine qva_menu is a recursive menu which allows interactive   */
  /* running of the QVA algorithm and allows the user to run QVA with   */
  /* different thresholds and number of iterations against the same     */
  /* scene without reading the monitor. lpic contains the raw pixel data*/
  /* from the left image and rpic contains the data from the right image*/
  /*************************************************************************/
  qva_menu(lpic, rpic)
  int lpic[250][250], rpic[250][250];
  {
   int  left,right,color;
   char qva_loop_flag[3],print_flag[3];
   int  i,j,k,l,m,n,fl;
   int  feature[250];          /* QVA feature vector                      */
   int  picture[250];          /* vector containing pixel color data      */
   int  lrqvapic[250][250];    /* image after left-right QVA smoothing    */
   int  tdqvapic[250][250];    /* image after top-down QVA smoothing      */
   int  qva_left[250][250];    /* results of applying QVA to left image   */
   int  qva_right[250][250];   /* results of applying QVA to right image */

   for(;;)
   {
     cls();
     printf("\n            ******** LEFT RIGHT COMPARISON TESTS ********");
     printf("\n\n\n                 0: Quit the test loop to VMS");
     printf("\n               1: Display results of applying QVA");
     printf("\n\n\n\n\n\n Select an option by its number\n>");
     scanf("%d",&option);

     switch (option)
     {
      case 0: return;

      case 1: cls();
              printf("\n Enter noise threshold\n>");
              scanf("%d",&thrsh);

              for(j=0; j<250; j++)         /* set up left image for QVA */
              {
               for(i=0; i<250; i++)  tdqvapic[i][j] = lpic[i][j];
              }
              text(40,390,0,2,0,"Left image");
              qva_loop_flag[0] = 'y';               /* set loop flag on */
              while(qva_loop_flag[0]=='Y' || qva_loop_flag[0]=='y')
```

```
                {
                 left_right(picture,feature,lrqvapic,tdqvapic);
                 top_down(picture,feature,lrqvapic,tdqvapic);
                 printf("\n\n\n Run QVA against left image again? (y/n)  ");
                 scanf("%s",qva_loop_flag);
                }
                for(j=0; j<250; j++)      /* save left image qva results */
                {
                 for(i=0; i<250; i++)  qva_left[i][j] = tdqvapic[i][j];
                }

                aclear(0,0,511,511,100);
                printf("\n\n\n\n\n\n");
                printf("Please wait. Right image set up in progress.... ");
                for(j=0; j<250; j++)       /* set up right image for QVA */
                {
                 for(i=0; i<250; i++)
                 {
                  tdqvapic[i][j] = rpic[i][j];
                  wpixel(i+1,j+1,rpic[i][j]);
                 }
                }
                text(40,390,0,2,0,"Right image");
                qva_loop_flag[0] = 'y';                /* set loop flag on */
                while(qva_loop_flag[0]=='Y' || qva_loop_flag[0]=='y')
                {
                 left_right(picture,feature,lrqvapic,tdqvapic);
                 top_down(picture,feature,lrqvapic,tdqvapic);
                 printf("\n\n\n Run QVA against right image again? (y/n)  ");
                 scanf("%s",qva_loop_flag);
                }
                for(j=0; j<250; j++)      /* save right image qva results */
                {
                 for(i=0; i<250; i++)  qva_right[i][j] = tdqvapic[i][j];
                }

                aclear(0,0,511,511,100);  /* print QVA results to monitor */
                for(j=0; j<250; j++)
                {
                 for(i=0; i<250; i++)
                 {
                  wpixel(i+1,j+1,qva_left[i][j]);
                  wpixel(i+261,j+1,qva_right[i][j]);
                 }
                }
                printf("\n\nEnter name of file\n>");
                scanf("%s",&test);
                saveim(1,1,510,250,0,test,"none");
                break;
    }
  }

return;
}
```

```
/*********************************************************************/
/*  Subroutine left_right processes the image data by columns from  */
/*  left to right.  Each column is assigned to a picture vector and */
/*  input to the QVA feature processing subroutines.  The feature   */
/*  vectore control the amount of brightness smoothing performed.   */
/*  Data resulting from the top_down subroutine (tdqvapic) is       */
/*  transformed into lrqvapic.  Pixel brightness averaging is used  */
/*********************************************************************/

left_right(picture, feature, lrqvapic, tdqvapic)
int picture[], feature[], lrqvapic[250][250], tdqvapic[250][250];
{
   int  i,j,left,right,color,color_avg,smooth_length;

   printf("\n\nprocessing left-right QVA feature vectors.... ");
   for(j=0; j<250; j++)
   {
      for(i=0; i<250; i++)  picture[i] = tdqvapic[i][j];
      create_feature(picture,feature);
      qva_rules(picture,feature);

      color = left = right = 0;
      while (right<249)
      {
         right = right + 1;
         if (feature[right]==1 || feature[right]==2 || right==249)
         {
            color = 0;
            smooth_length = right - left;
            for(i=left; i<right; i++) color = color + tdqvapic[i][j];
            color_avg = color / smooth_length;
            for(i=left; i<right; i++) lrqvapic[i][j] = color_avg;
            left = right;
         }
      }
   }

   for(j=1; j<251; j++)    /* print left-right results to the monitor */
   {
      for(i=261; i<511; i++)   wpixel(i,j,lrqvapic[i-261][j-1]);
   }

return;
}
```

```
/*********************************************************************/
/*  Subroutine top_down processes the image data by rows from        */
/*  left to right.  Each row is assigned to a picture vector and     */
/*  input to the QVA feature processing subroutines.  The feature    */
/*  vectore control the amount of brightness smoothing performed.    */
/*  Data resulting from the left_right subroutine (lrqvapic) is      */
/*  transformed into tdqvapic.  Pixel brightness averaging is used   */
/*********************************************************************/

top_down(picture, feature, lrqvapic, tdqvapic)
int picture[], feature[], lrqvapic[250][250], tdqvapic[250][250];
{
   int  i,j,left,right,color,color_avg,smooth_length;

   printf("\nprocessing top-down QVA feature vectors.... ");
   for(i=0; i<250; i++)
   {
      for(j=0; j<250; j++)  picture[j] = lrqvapic[i][j];
      create_feature(picture,feature);
      qva_rules(picture,feature);

      color = left = right = 0;
      while (right<249)
      {
         right = right + 1;
         if (feature[right]==1 || feature[right]==2 || right==249)
         {
            color = 0;
            smooth_length = right - left;
            for(j=left; j<right; j++) color = color + lrqvapic[i][j];
            color_avg = color / smooth_length;
            for(j=left; j<right; j++) tdqvapic[i][j] = color_avg;
            left = right;
         }
      }
   }

   for(j=261; j<511; j++)   /* print top-down results to the monitor */
   {
      for(i=261; i<511; i++)   wpixel(i,j,tdqvapic[i-261][j-261]);
   }

return;
}
```

```
/*********************************************************************/
/*  Subroutine create_feature uses the video data in the picture     */
/*  vector and creates a feature vector corresponding to those       */
/*  pixels.  The threshold entered by the user is a global variable  */
/*  used to determine the type of feature assigned to each pixel.    */
/*  Processes vector of length 250.  Either row or column data       */
/*********************************************************************/
create_feature(picture, feature)
int  picture[], feature[];
{
   int  dif,i;

   for(i=1; i<250; i++)       /* difference between succesive pixels */
   {
    dif = picture[i-1] - picture[i];
    if     (dif < -thrsh)              feature[i] = 2;   /* pos edge */
    else if(-thrsh <= dif && dif < 0)  feature[i] = 4;   /* pos grad */
    else if(dif == 0)                  feature[i] = 5;   /* smooth   */
    else if(0 < dif && dif <= thrsh)   feature[i] = 3;   /* neg grad */
    else                               feature[i] = 1;   /* neg edge */
   }
return;
}
```

```
/*********************************************************************/
/* Subroutine qva_rules applies QVA production rules to the feature  */
/* vectors. Input feature are analyzed and all gradients are removed */
/* leaving only edges and smooth pixel brightness features.          */
/* picture data and threshold used to control the amount of smoothing*/
/* Processes vector of length 250.  Either row or column data        */
/*********************************************************************/
qva_rules(picture,feature)
int  picture[], feature[];
{
   int  left, right, temp, length;

   right = left = 0;
   while (right < 250)
   {
      right = right + 1;
      if(abs(picture[right] - picture[left]) > thrsh
         || feature[right]==1 || feature[right]==2)
      {
         length = right - left;          /* how far from last edge */
         if (length = 1)
         {
            if (feature[right]==3)  feature[right] = 1;
            else                    feature[right] = 2;
            left = right;
         }
         else if (length = 2 && feature[left]==1 && feature[right]==1)
```

```
            {
               feature[left+1]=1;
               left = right;
            }
            else if (length = 2 && feature[left]==2 && feature[right]==2)
            {
               feature[left+1]=2;
               left = right;
            }
            else
            {
               temp = right;
               if (feature[temp]==1 && feature[temp-1]==3)
               {
                  temp = temp - 1;
                  while (feature[temp]==feature[temp-1] && temp>left)
                  {
                     temp = temp - 1;
                  }
                  temp = temp + 1;
               }
               else if (feature[temp]==2 && feature[temp-1]==4)
               {
                  temp = temp - 1;
                  while (feature[temp]==feature[temp-1] && temp>left)
                  {
                     temp = temp - 1;
                  }
                  temp = temp + 1;
               }
               if(feature[temp]==3)  feature[temp] = 1;
               else                  feature[temp] = 2;

               left = left + 1;           /* start smoothing after edge */
               while (left < temp)
               {
                  feature[left] = 5;
                  left = left + 1;
               }
               left = temp;                        /* reset left pointer */
      } } }
   return;
   }


   /*************************************************************************/
   /*  Subroutine used as a clear screen during interactive opperation  */
   /*************************************************************************/
   cls()
   {
     printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
   return;
   }

   /*************************   END OF LR_QVA.C   *************************/
```

```
/*************************************************************************
*                                                                        *
*       Program Name   DIST_COMPARE.C                                    *
*                                                                        *
*       Author : Richard E. Roberts Aug 1987                             *
*                                                                        *
*   Special thanks to Larry Lambert for his assistance in developing     *
*   and debugging the image comparison and distance algorithms used.     *
*                                                                        *
*   This program identifies regions in the left image, matches those     *
*   regions in the right image, and computes the distance to them.       *
*   The images are input to the program from the video monitor.          *
*   The left image must be located in position x=0 y=0 to x=249 y=249.   *
*   while the right image must be located at x=260 y=0 to x=511 y=249.   *
*************************************************************************/

#include "sys$library:stdio.h"
#include "sys$library:math.h"
#include "dua0:[iti100.itex]stdtyp.h"
#include "dua0:[iti100.itex]itex100.h"

/*************************************************************************/
/*                      GLOBAL VARIABLES                                 */
/*************************************************************************/
static int   option;
int          qva_loop;
int          qva_left[250][250], qva_right[250][250];
int          mask_array[250][250], pattern_array[250][250];
int          dist_result[100] = 0;


/*************************************************************************/
/*                       START MAIN PROGRAM                              */
/*  QVA processed image data is input from the video monitor             */
/*************************************************************************/
main()
{

/*   Initialize ITEX100 board and monitor   */
unsigned        base = 0x1600;
long            mem = 0x200000L;
int             flag = 1, block = 8;
int             errnum;

int     i,j,k,l,m,n, fl;
int     xlft, xrgt, ytop, ybot, color;
int     a_size;
int     pixarray[512];
int     total = 0;
sethdw(base, mem, flag, block);
```

```
cls();
printf("\n       ******  LEFT RIGHT IMAGE MATCH TEST   ******");
printf("\n\n\n\n\n\n\n\n\n  Please wait while left and right");
printf(" images are separated and stored....\n\n\n ");
for(j=0; j<250; j++)          /* obtain left and right images from   */
{                                        /* monitor video RAM memory   */
   rhline(0,j,512,pixarray);
   for(i=0; i<250; i++)
   {
      qva_left[i][j]  = pixarray[i];    /* read left image into array */
      qva_right[i][j] = pixarray[i+260];          /* read right image */

      mask_array[i][j] = 0;          /* initialize all mask pixels off */
} }

carea(1,1,250,250,1,261,250,250);
line(60,0,60,250,0);           /* left limit of left image search area */
line(450,0,450,250,0);       /* right limit of right image search area */
line(0,0,0,512,0);




/*************************************************************************/
/* Now find "good" objects in left image.  Start with large objects   */
/* of size 41 X 41 and decrease the size of the minimum object each   */
/* time until object size is 11 X 11.  Search area constrained to     */
/* x=80 thru x=250 because left image contains information not        */
/* contained in right image.  Similiarily the right 50 pixels of the  */
/* right image contain information not visible in left image.         */
/*************************************************************************/

for(a_size=20; a_size>=5; a_size=a_size-5)
{
  printf("\n\n\nPattern area search size is");
  printf(" %2d by %2d",2*a_size+1,2*a_size+1);
  printf("\n\n\n\nLooking for good object in left image...");
  for(j=a_size; j<240-a_size; j+=(a_size/2))   /* large objects first */
  {
    for(i=60; i<250; i+=2)   /* limit search start area in left image */
    {
      fl = 0;                                      /* initialize flag off */
      for(l=j-a_size; l<=j+a_size; l++)
        for(k=i-a_size; k<i+a_size; k++)
          if(qva_left[k][l] != qva_left[k+1][l])   fl = 1;

      for(l=j-a_size; l<=j+a_size; l++)
        if (qva_left[k][l]!=qva_left[k][l+1] || rpixel(k,l)==0) fl = 1;

      if (fl == 0)   /* if flag off -> object found with minimum size */
      {
        total += 1;
        flood(k,l,255);
        xrgt = ybot = 0;
        xlft = ytop = 250;
```

```
        for(n=0; n<250; n++)           /* find the size of the object */
        {
          for(m=0; m<250; m++)
          {
            color = rpixel(m,n);
            if(color==255 && m>xrgt)    xrgt=m;
            if(color==255 && n>ybot)    ybot=n;
            if(color==255 && m<xlft)    xlft=m;
            if(color==255 && n<ytop)    ytop=n;
        } }
        if(xlft < 60) xlft = 60;     /* truncate area with over flood */

        digit(xlft,ytop+261,xrgt-xlft,ybot-ytop,total);
        for(n=ytop; n<=ybot; n++)
        {
          for(m=xlft; m<=xrgt; m++)
          {
            color = rpixel(m,n);
            if (color==255)  pattern_array[m][n] = qva_left[m][n];
            else             pattern_array[m][n] = 0;
        } }
        flood(k,1,0); /* indicate identified objects with black color */

        find_match(xlft, xrgt, ytop, ybot, a_size, total);

        printf("\n\n\n\nLooking for good object in left image...");
    } } }

}    /****   END OF LOOP TO SELECT REGIONS IN THE LEFT IMAGE  ****/


/*************************************************************************/
/*    FORMAT RESULTS ON THE VIDEO MONITOR FOR READABILITY                */
/*************************************************************************/
aclear(0,0,252,252,100);
carea(1,261,250,250,1,1,250,250);
aclear(1,260,252,252,100);

line(255,255,255,480,0);
line(251,430,255,430,0);
number(230,425,50);
line(251,380,255,380,0);
number(230,375,100);
line(251,330,255,330,0);
number(230,325,150);
line(251,280,255,280,0);
number(230,275,200);
line(251,480,255,480,0);
number(230,475,0);

printf("\n\n\n\nOBJECT   DISTANCE");
for(i=1; i<=total; i++)   printf("\n  %d       %d",i,dist_result[i]);

}          /******   END OF MAIN PROGRAM LOOP   *****/
```

```
/*************************************************************************/
/* Find corresponding object in right image to match object              */
/* selected in left image                                                */
/*                                                                       */
/* Search rectangular areas of right image for matching object           */
/* Size of search area supplied by main program                          */
/*************************************************************************/

find_match(xlft, xrgt, ytop, ybot, a_size, total)
int  xlft, xrgt, ytop, ybot, a_size, total;
{
  int    i, j, k, l, m, n, color;
  int    mask_flag, x_index;
  int    small_x, small_y, x_start, y_start, x_size, y_size, x_stop;
  int    middle_x, middle_y, tst_val, n_offset, m_offset;
  double dif[250][250], value, testpt1, testpt2;
  double huge_num;

  printf("\nLooking for matching object in right image...");
  x_size = xrgt - xlft;
  y_size = ybot - ytop;
  middle_x = x_size / 2;
  middle_y = y_size / 2;
  huge_num = (double) (256.0 * 250.0 * 250.0);

  if (ytop > 5)   y_start = ytop-2;      /* limit size of search area */
  else            y_start = 0;
  for(l=y_start; l<ytop+13; l++)         /* manually measured offsets */
  {
    for(k=0; k<200; k++)                 /* manually measured offsets */
    {
      mask_flag = 0;                      /* initialize mask flag off */
      tst_val = 0;
      dif[k][l] = 0;
      for(n=0; n<=y_size; n++)
      {
        color = pattern_array[middle_x+xlft][n+ytop];
        if(color != 0)
           tst_val=tst_val+abs(color-qva_right[middle_x+k][n+l]);
        if(mask_array[middle_x+k][n+l] == 100)  mask_flag = 1;
      }
      for(m=0; m<=x_size; m++)
      {
        color = pattern_array[m+xlft][middle_y+ytop];
        if(color != 0)
           tst_val=tst_val+abs(color-qva_right[m+k][middle_y+l]);
        if(mask_array[m+k][middle_y+l]==100||(m+k)>200) mask_flag=1;
      }
      if(tst_val>(x_size+y_size)*10 || mask_flag==1)
            dif[k][l] = huge_num;
      else
      {
        for(n=0; n<=y_size; n++)
        {
```

```
            for(m=0; m<=x_size; m++)
            {
              color = pattern_array[m+xlft][n+ytop];
              if (color != 0)  value=abs(color-qva_right[m+k][n+l]);
              else             value=0;
              dif[k][l] = dif[k][l] + value;
              if(mask_array[m+k][n+l] == 100)  mask_flag = 1;
          } }
          if(mask_flag == 1)   dif[k][l] = huge_num;
        }
    } }

    small_x = 0;                                    /* initialize variables */
    small_y = y_start;
    testpt1 = dif[0][y_start];
    testpt2 = 0;
    for(l=y_start; l<ytop+13; l++)
    {
      for(k=0; k<200; k++)
      {
        if(dif[k][l] < testpt1)
        {
          testpt2 = testpt1;
          testpt1 = dif[k][l];
          small_x = k;
          small_y = l;
    } } }
    printf("\n\nLowest correlation difference = %f",testpt1);
    printf("\nNext lowest correlation difference = %f",testpt2);




/*************************************************************************/
/* If the top two correlation differences are withen 50 of each other */
/* then the distances are too close to acurately select a matching    */
/* region in the image.   By increasing the size of the window, extra */
/* information is available to match the regions.                     */
/*************************************************************************/
    if (testpt2 <= testpt1 + 40)
    {
      printf("\n\nUsing larger window to find right image object...");
      if (ytop > 5)   y_start = ytop-2;
      else            y_start = 0;

      switch (a_size) {     /* accelerate matching by constraining area */
      case 5:  if(xlft >= 60)    x_start = xlft - 60;
               else              x_start = 0;
               x_stop = xrgt-20;
               x_index = 1;
               break;
```

```
case 10: if(xlft >= 100)   x_start = xlft - 100;
         else              x_start = 0;
         x_stop = xrgt;
         x_index = 1;
         break;
case 15: if(xlft >= 150)   x_start = xlft - 150;
         else              x_start = 0;
         x_stop = 200;
         x_index = 2;
         break;
default: x_start = 0;
         x_stop = 200;
         x_index = 2;
         break;
}

for(l=y_start; l<ytop+13; l++)       /* manually measured offsets */
{
  for(k=x_start; k<x_stop; k+=x_index)
  {                                  /* manually measured offsets */
    mask_flag = 0;                    /* initialize mask flag off */
    tst_val = 0;
    dif[k][l] = 0;
    for(n=3; n<=y_size-3; n++)             /* allow small overlap */
    {
      tst_val = tst_val + abs(qva_left[middle_x+xlft][n+ytop]
                            - qva_right[middle_x+k][n+l]);
      if(mask_array[middle_x+k][n+l] == 100)    mask_flag = 1;
    }
    for(m=3; m<=x_size-3; m++)             /* allow small overlap */
    {
      tst_val = tst_val + abs(qva_left[m+xlft][middle_y+ytop]
                            - qva_right[m+k][middle_y+l]);
      if(mask_array[m+k][middle_y+l]==100||(m+k)>200) mask_flag=1;
    }
    if(tst_val>(x_size+y_size-12)*20 || mask_flag==1)
          dif[k][l] = huge_num;
    else
    {
      if(ytop > 4) n_offset = 4;  /* keep array withen x,y bounds */
      else         n_offset = 4 - ytop;
      if(xlft > 4) m_offset = 4;
      else         m_offset = 4 - xlft;

      for(n=0; n<=y_size+8; n++)   /* increase height by 8 pixels */
      {
        for(m=0; m<=x_size+8; m++) /* increase length by 8 pixels */
        {
          value = abs(qva_left[m+xlft-m_offset][n+ytop-n_offset]
                    - qva_right[m+k-m_offset][n+l-n_offset]);
          dif[k][l] = dif[k][l] + value;
} } } } }
```

```
    small_x = x_start;                /* initialize boundary variables */
    small_y = y_start;
    testpt1 = dif[x_start][y_start];
    testpt2 = 0;
    for(l=y_start; l<ytop+13; l++)
    {
      for(k=x_start; k<x_stop; k=k+x_index)
      {
        if(dif[k][l] < testpt1)
        {
          testpt2 = testpt1;
          testpt1 = dif[k][l];
          small_x = k;
          small_y = l;
    } } }
    printf("\nLowest correlation difference = %f",testpt1);
    printf("\nNext lowest correlation difference = %f",testpt2);
  }




/*************************************************************************/
/* First draw a  rectangle around matching region in right image         */
/* Then compute the distance using the matching regions.                 */
/* If no regions are matched then distances are not calculated.          */
/*************************************************************************/
  if(testpt1!=0 && testpt1!=huge_num)           /* is position valid */
  {
    digit(small_x+260,small_y,x_size,y_size,total);
    distance(xlft, small_x, x_size, total);/* find distance to object */
    for(l=0; l<=y_size; l++)
    {
      for(k=0; k<=x_size; k++)
      {
        if (pattern_array[k+xlft][l+ytop] != 0)
                        mask_array[k+small_x][l+small_y] = 100;
  } } }

return;
}              /****  END OF FIND_MATCH SUBROUTINE  ****/
```

```
/*************************************************************************/
/*  Calculate the distances to the objects identified in both images  */
/*                                                                       */
/*  Left image object   x=xlft      y=ytop      dx=x_size   dy=y_size */
/*  Right image object  x=x_small   y=y_small   dx=x_size   dy=y_size */
/*  xl = left image objects top left corner position measured from 0  */
/*  xr = right image objects top left corner position measured from 0 */
/*                                                                       */
/*  Angle conversion factor from # of pixels to # of degrees          */
/*  1.592deg = 22 pixels at 72 in camera seperation from object       */
/*                                                                       */
/*  Seperation distance between cameras   cam_sep = 9.0 inches        */
/*                                                                       */
/*************************************************************************/
#define pi    3.14159
#define cam_sep  9.0          /* camera seperation distance in inches */

distance(xl, xr, xsize, count)
int  xl, xr, xsize, count;
{

  int     i, color;
  float   temp1, temp2, t1, t2;
  double  theta1, theta2, theta3, reslt1, reslt2, reslt3;
  double  z_distance;
  double  sin();

  for(i=0; i<=xsize; i++)
  {
    t1 = i + xl;                      /* left image object displacement */
    if (t1 < 125.0) temp1 = 90.0 + ((125.0 - t1) * (1.592/22.0));
    else            temp1 = 90.0 - ((t1 - 125.0) * (1.592/22.0));
    t2 = i + xr;                     /* right image object displacement */
    if (t2 > 125.0) temp2 = 90.0 + ((t2 - 125.0) * (1.592/22.0));
    else            temp2 = 90.0 - ((125.0 - t2) * (1.592/22.0));

    theta1 = temp1 * (pi/180.0);          /* convert angle to radians */
    theta2 = temp2 * (pi/180.0);
    reslt1 = sin(theta1);
    reslt2 = sin(theta2);
    theta3 = (180.0 - (temp1 + temp2)) * (pi/180.0);
    reslt3 = sin(theta3);
    if (reslt3 < 0.0) reslt3 = reslt3 * -1.0;       /* absolute value */
    if(reslt3==0.0) printf("\n\n\nreslt3 = 0. cannot divide by 0.");
    z_distance = (cam_sep * reslt1 * reslt2) / reslt3;
    color = (int) z_distance;         /* convert to integer for output */

    wpixel(i+xr+260, 480-color, 0);
  }

  dist_result[count] = color; /* save the distance calculation result */
  num(xr+(xsize/2)+260,480-color-4,count);/*number each distance line */

return;
}               /****  END OF DISTANCE SUBROUTINE  ****/
```

```
/*************************************************************************
*                                                                        *
*     Subroutine name    DIGIT                                          *
*                                                                        *
*       Author :     Larry Lambert  Sept 1987                            *
*       Modified by: Richard Roberts                                     *
*       Printd numbers between 0 and 99 inside region rectangles         *
*************************************************************************/
static int  data[10][28] =
    {
       {0,0,0,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,0,0,0},
       {1,0,1,1,0,0,1,1,1,0,1,1,1,0,1,1,1,0,1,1,0,1,1,0,0,0,1,0,0,0,1},
       {0,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0,0,1,1,1,0,1,1,1,0,0,0,0},
       {0,0,0,0,1,1,0,1,1,0,1,1,0,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0},
       {0,1,1,1,0,1,0,1,0,1,0,1,0,0,0,0,1,1,0,1,1,1,0,1,1,1,0,1},
       {0,0,0,0,0,1,1,1,0,1,1,1,0,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0},
       {0,0,1,1,0,1,1,1,0,1,1,1,0,0,0,0,0,1,1,0,0,1,1,0,0,0,0,0},
       {0,0,0,0,0,1,1,0,1,1,1,0,1,1,1,0,1,1,1,0,1,1,1,0,1,1,1,0},
       {0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,1,1,0,0,1,1,0,0,0,0,0},
       {0,0,0,0,0,1,1,0,0,1,1,0,0,0,0,0,1,1,1,0,1,1,1,0,1,1,1,0}
    };

digit(xlft, ytop, x_size, y_size, value)
int  xlft, ytop, value;
{
  int i, j, num1, num2, color;

  num1 = value / 10;                                    /* first digit */
  num2 = value - (10*num1);                             /* second digit */
  color = rpixel(xlft+3, ytop+3);                  /* sample color test */
  if(color < 80) rectangle(xlft,ytop,x_size,y_size,200);
  else           rectangle(xlft,ytop,x_size,y_size,0);
  xlft +=1;                      /* starting point to print first digit */
  ytop +=1;
  for(i=0; i<=3; i++)                               /* print first digit */
  {
    for(j=0; j<=6; j++)
    {
      if(data[num1][i+(j*4)] == 0)
      {
        if(color < 80) wpixel(xlft+i, ytop+j, 200);
        else           wpixel(xlft+i, ytop+j, 0);
  } } }

  for(i=0; i<=3; i++)                              /* print second digit */
  {
    for(j=0; j<=6; j++)
    {
      if(data[num2][i+(j*4)] == 0)
      {
        if(color < 80) wpixel(xlft+5+i, ytop+j, 200);
        else           wpixel(xlft+5+i, ytop+j, 0);
  } } }
return;
}           /****   END OF DIGIT SUBROUTINE   ****/
```

```
/*************************************************************************
*                                                                        *
*     Subroutine Name    NUMBER                                         *
*                                                                        *
*       Author :     Larry Lambert  Sept 1987                            *
*       Modified by: Richard Roberts                                     *
*       Prints numbers between 0 and 250 on distance line scale          *
*************************************************************************/

number(xlft, ytop, value)
int  xlft, ytop, value;
{
  int i, j, num1, num2, num3;

  num1 = value / 100;                                    /* first digit */
  num2 = (value - (100*num1)) / 10;                     /* second digit */
  num3 = value - ((100*num1) + (10*num2));               /* third digit */

  xlft +=1;                                  /* starting point for digits */
  ytop +=1;
  for(i=0; i<=3; i++)                                    /* print first digit */
    for(j=0; j<=6; j++)
      if(data[num1][i+(j*4)] == 0)   wpixel(xlft+i, ytop+j, 0);

  for(i=0; i<=3; i++)                                   /* print second digit */
    for(j=0; j<=6; j++)
      if(data[num2][i+(j*4)] == 0)  wpixel(xlft+5+i, ytop+j, 0);

  for(i=0; i<=3; i++)                                    /* print third digit */
    for(j=0; j<=6; j++)
      if(data[num3][i+(j*4)] == 0)  wpixel(xlft+10+i, ytop+j, 0);

return;
}



/*************************************************************************/
/*    SUBROUTINE: NUM     FUNCTION: prints numbers on distance lines  */
/*************************************************************************/
num(xlft, ytop, value)
int  xlft, ytop, value;
{
  int i, j, num1, num2;

  num1 = value / 10;                                     /* first digit */
  num2 = value - (10*num1);                             /* second digit */

  xlft +=1;                                  /* starting point for digits */
  ytop +=1;
  for(i=0; i<=3; i++)                                    /* print first digit */
    for(j=0; j<=6; j++)
      if(data[num1][i+(j*4)] == 0)   wpixel(xlft+i, ytop+j, 200);
```

```
  for(i=0; i<=3; i++)                              /* print second digit */
    for(j=0; j<=6; j++)
      if(data[num2][i+(j*4)] == 0)  wpixel(xlft+5+i, ytop+j, 200);

return;
}


/*************************************************************************/
/*  Subroutine used as a clear screen during interactive opperation   */
/*************************************************************************/
cls()
{
   printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");

return;
}
```

Bibliography

[Aut85] Automation Technology Corporation. Final Report of Phase I SBIR Contract DAAE07-85-C-R083, 3-D Viewing System Enhancements For The Control of Robotic Vehicles, Technical Report, No. ARD-5463B, Columbia MD, Aug 1985 (AD-B107 207).

[Bli85] Blicher, Peter A. Edge Detection and Geometric Methods in Computer Vision, Technical Report, No. STAN-CS-85-1041, February 1985.

[Hol85a] Holten, James R., Rogers Steven K., Kabrisky Matthew, and Cross Steven, "Stereo Image Ranging for an Aoutonomous Robot Vision System", Proceedings of the SPIE International Conference on Intelligent Robots and Computer Vision, Cambridge MA, September 1985.

[Hol85b] Holten, James R. A Robot Vision System, PhD dissertation AFIT/DS/ENG/85D-1. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1985 (AD-A164 202).

[Hol87] Holten, James R. Personal communication, 15 July 1987.

[Hor86] Horn, Berthold Klaus Paul. Robot Vision. The MIT Electrical Engineering and Computer Science Series, McGraw-Hill Book Company, New York, 1986.

[Ite86] Imaging Technology Incorporated, ITEX 100 Programmer's Manual, Technical Publications Department, 600 West Cummings Park, Woburn MA, May 1986.

[Kab87] Kabrisky, Matthew, Professor Electrical Engineering. Personal interview, Air Force Institute of Technology, Wright-Patterson AFB, OH, April 1987.

[Mor83] Moravec, Hans P. "The Stanford Cart and the CMU Rover," Proceedings Of The IEEE, Vol 71, No. 7: 872 - 884, July 1983.

[Pou86] Poulos, Dennis D. Range Image Processing For Local Navigation Of An Autonomous Land Vehicle, MS Thesis, Naval Postgraduate School, Monterey, CA, Sept 1986 (AD-A171 053).

VITA

Richard E. Roberts was born on 3 May 1957 in Miami Beach, Florida. He graduated from Pace High School in Opa-Locka, Florida in 1975 and attended the University of Florida in Gainesville, Florida from which he received the degree Bachelor of Science in Computer and Information Sciences in June 1980. He entered the United States Air Force on active duty in May 1981 and received his commission from Officer Training School in August 1981. From August 1981 until June 1983 he attended the University of South Florida in Tampa, Florida where he received the degree Bachelor of Science in Electrical Engineering. After graduation he was assigned to the Shuttle Activation Task Force at Vandenberg AFB, California as a instrumentation systems engineer. He entered the Masters Program in the School of Engineering, Air Force Institute of Technology, in June 1986.

Permanent Address: 1235 N.W. 128 Street
N.Miami, Florida 33167

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b RESTRICTIVE MARKINGS: N/A
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT: Approved for Public Release: Distribution Unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT / GE / ENG / 87D-54	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b OFFICE SYMBOL (If applicable): AFIT/ENG	7a NAME OF MONITORING ORGANIZATION
6c ADDRESS (City, State, and ZIP Code): Air Force Institute of Technology, Wright-Patterson AFB, OH. 45433	7b ADDRESS (City, State, and ZIP Code)
8a NAME OF FUNDING/SPONSORING ORGANIZATION; 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
Three-Dimensional Scene Analysis Using Stereo Based Imaging

12 PERSONAL AUTHOR(S)
Roberts, Richard E., Captain, USAF

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Masters Thesis	FROM _____ TO _____	87/12/09	127

16 SUPPLEMENTARY NOTATION

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP
12	09	

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Pattern Recognition; Artificial Intelligence; Computer Vision; Robotics; Image Processing

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Thesis Chairman: Matthew Kabrisky, PhD
Professor of Electrical Engineering

Approved for public release IAW AFR 190-1.
[illegible] 4 Mar 88
[illegible] Development
Air [illegible]
Wright [illegible]

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL: Dr. Matthew Kabrisky, AD-24	22b TELEPHONE (Include Area Code): (513) 255-5276; 22c OFFICE SYMBOL: AFIT/ENG

DD Form 1473, JUN 86 *Previous editions are obsolete*

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

continued from block 19 - ABSTRACT:

This thesis presents a new method for using passive binocular vision to create a map of the top-view of a robot's environment. While numerous autonomous robot navigation systems exist, most attempt to match objects in each image by following edges or locating significant groups of edge pixels. The method described in this paper uses two cameras (alligned in parallel) to generate stereo images. Low level features are extracted using a new non-linear production rule system, rather than a conventional filter design. The features are registered by matching correspondingly shaped regions of constant brightness levels in both images and the offsets are then computed. The use of heuristics to relieve the computational burden associated with low level image processing is unique; both in processing the images and in locating matching regions in the images. The feature extraction algorithm, the intermediate symbolic representations, and the application of these results to hierarchical structures common to context queuing systems are presented.

END

DATE

FILMED

7-88

DTIC