

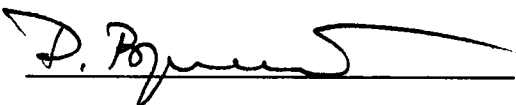
To the Graduate Council:

I am submitting herewith a thesis written by Charles Henry Hayden Jr. entitled "A Temporal Edge-Based Segmentor." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.



R.C. Gonzalez, Major Professor

We have read this thesis
and recommend its acceptance:







Vice Provost
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature

Charles D. Dwyer /

Date

11.12.85

A TEMPORAL EDGE-BASED SEGMENTOR

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Charles H. Hayden Jr.

December 1985

ACKNOWLEDGMENTS

The author wishes to thank all those who aided in the research and preparation of this thesis. I would like to particularly thank Darryl Cate and John Wright for their assistance with the facilities at Texas Instruments in Dallas and Bobby Bodenheimer and Jenny Craven for their help in the final preparation and formatting of this document. I extend special thanks to Dr. R.C. Gonzalez, who has been a source of assistance and inspiration throughout my graduate career.

Also, I sincerely thank Texas Instruments for their financial support through the Image Processing Fellowship at The University of Tennessee and use of their research facilities in Dallas. I am indebted to Dr. Bill Eversole for his help in coordinating my relationship with Texas Instruments.

ABSTRACT

An edge-based segmentation algorithm is developed by enhancing a simple edge detection method with temporal information. A brief overview of segmentation, with an emphasis on edge detection methods, is presented before discussing the theory and implementation of the Temporal Edge-Based Segmentor (TEBS). The edge detection method used in TEBS is described in detail along with the methods that implement temporal information. The performance is discussed along with suggestions for improvement and further implementation.

TABLE OF CONTENTS

CHAPTER	PAGE
I INTRODUCTION	1
II SURVEY OF SEGMENTATION RESEARCH	3
2.1 FEATURE-BASED METHODS	3
2.2 REGION-BASED METHODS	4
2.3 EDGE-BASED METHODS	5
III TEBS ALGORITHM	9
3.1 EDGE DETECTION	9
3.2 TEMPORAL INFORMATION	13
IV RESULTS	26
4.1 SYNTHETIC IMAGES	26
4.2 FLIR DATA	27
V CONCLUSIONS AND RECOMMENDATIONS	37
BIBLIOGRAPHY	39
APPENDICES	43
A. ALPHA FILTER NOISE REDUCTION	44
B. TEBS PROGRAM AND SUBROUTINES	45
VITA	84

LIST OF FIGURES

FIGURE	PAGE
2.1 Robert's, Sobel, and Prewitt Gradient Operators	7
3.1 3x3 Pixel Cell for Prewitt Gradient	10
3.2 Original Image and Edge Magnitude Image	12
3.3 Thinned Edges and Thresholded Edges	12
3.4 Block Diagram of Alpha Filter	14
3.5 Feedback Coefficient vs. Number of Frames	16
3.6 Noise Reduction vs. Number of Frames	16
3.7 Edges Extracted from Sequential Frames	19
3.8 Temporal Information in Midprocessing	20
3.9 Guided Threshold Relaxation Process	22
3.10 Directional Mask for Single-Frame Postprocessing	23
3.11 Temporal Information in Postprocessing	25
4.1 Filtering and Edges Extraction with Synthetic Data	28
4.2 Edge Linking and Object Segmentation with Synthetic Data .	29
4.3 Original Scene Image	31
4.4 Extracted Subimage and Filtered Subimage	32
4.5 Edge Magnitude Image and Thinned Edge Image	32
4.6 Thresholded Edges	33
4.7 Thick Edges	33
4.8 Relaxed Edges	33
4.9 Cleaned Edges and Preserved Edges	34
4.10 Segmented Object	34

CHAPTER I

INTRODUCTION

Object recognition is an important application of image processing. The basis for any automated recognition system is a lower level process called segmentation. Segmentation can be defined as a decision process with the objective of establishing boundaries between regions. Segmentation is a problem that has not yet been "solved." Although automation has been introduced into many segmentation approaches, there is a significant trade-off between generalization and performance. In order to take steps toward solving the problem, methods must be discovered that make better use of information already in the image.

One type of information inherent to any image with a discernible object are its edges. Edge detection techniques are usually followed by combination techniques to link edge segments. Present edge combination techniques are limited by the edge information available in a single frame and rely on sophisticated analytical or heuristic methods to create edge pixels between edge segments. By using multi-frame techniques, edge combination can be performed in a more exact manner using real edge information available in other frames. This thesis develops multi-frame techniques to improve edge detection and combination by using temporal information in the phases of preprocessing, midprocessing, and postprocessing of a simple edge detection method.

A survey of research on current methods of segmentation is presented in Chapter II. The topics of edge detection and edge linking are of special interest and are discussed in detail. Chapter III explains the approach used

in detecting object boundaries. This chapter also discusses in detail the use of temporal information to improve edge detection and provide edge combination. The results of these methods as applied to both synthetic and real data are compiled in Chapter IV. Discussion of the data reiterates the explanation of the methods explained in Chapter III by discussing the specific algorithms used to obtain the intermediate results during the segmentation process. Conclusions and suggestions for further research are discussed in Chapter V. Possible improvement and automation of these methods is also briefly discussed.

CHAPTER II

SURVEY OF SEGMENTATION RESEARCH

Over the past decade there has been increasing desire for automation in a variety of fields that require intelligent interpretation of images, such as medicine, weather analysis and warning, target recognition, and document reading. Image processing experts have addressed these problems and developed successful methods to perform these tasks. The extreme difficulty of these tasks and their predominately heuristic solutions have resulted in many specialized image processing techniques. Presently, almost all segmentation techniques are *ad hoc* in nature and will continue to be so until a general solution can be achieved. As a result, present segmentation techniques are almost innumerable. These techniques can, however, be divided into three main classes: feature-based, region-based, and edge-based. Feature and region-based methods are similar in nature and are discussed in general, while edge-based methods are discussed in detail.

2.1 FEATURE-BASED METHODS

Historically, feature-based segmentation methods have most commonly used intensity statistics. These methods have now been expanded to include many other features on both a local and global level. Statistical thresholding involves threshold selection based on the occurrence of features and can be divided into three major categories with the help of a threshold operator. The threshold operator is defined as a function T of the form

$$T(x, y, N(x, y), g(x, y))$$

where $g(x, y)$ represents the intensity of the point (x, y) and $N(x, y)$ quantifies some local property of the point (x, y) [33]. If T depends only on $g(x, y)$, it is considered a global thresholding function. When T depends only on $g(x, y)$ and $N(x, y)$, it is called a local threshold. If T depends on $x, y, g(x, y)$, and $N(x, y)$, the threshold is called dynamic. Present methods utilize features such as color intensities, hue, texture, geometry, co-occurrence matrices, and percent misclassification. An overview of applications using these methods can be found in survey papers by Fu [8] and Weszka [33].

By expanding the idea of statistical thresholding to its fullest potential, methods have been developed that use the pattern recognition process of clustering to segment an object from its background. Texture measures of a local neighborhood are popular features for clustering methods [3,4,35]. Many combinations of features have been implemented to attempt to achieve mutually exclusive clusters. Techniques have been developed using other pattern recognition methods such as syntactic techniques and tree automata. An overview of these applications is also found in the segmentation survey papers mentioned above [8,33].

Several disadvantages are inherent in segmentation methods based on features. Features are generally image dependent and restrict the techniques to specific applications. Not only are features specialized for each particular type of data but they are seldom consistent throughout noisy data.

2.2 REGION-BASED METHODS

Region-based methods can be viewed as an extension of feature-based methods. Features are used in region-based methods to merge or divide an image into separate regions. Methods of merging regions use neighbor

searches for similar properties such as intensity or texture to grow the region [19]. Other methods use semantics [7,34], relaxation [25], minimum distance classification [10,11], and linguistics [31] to perform region merging. Methods of region dividing use the mean vector of gray levels of multispectral image [23] or regular decomposition [15,16] to separate the image into regions.

A major disadvantage to region-based methods is the iterative nature of the techniques. As a result, region-based methods usually require extensive amounts of computation time and memory. Most algorithms also use local information extensively and, thus, have low noise tolerances.

2.3 EDGE-BASED METHODS

Edge-based segmentation techniques rely on detecting discontinuity points in the image and connecting them to obtain boundaries. An abrupt change of intensity in the image is considered to be characteristic of the border between regions. There are several motivating factors for using the edge detection approach in segmentation. A primary reason is that the majority of information existing in a image lies in the vicinity of the boundary between two regions. Also, biological visual systems appear to use edge detection instead of thresholding [26].

There are several methods of edge-based segmentation used in a variety of applications. Three of the more popular methods are: high-emphasis spatial frequency filtering, functional approximation, and gradient operators. Most edge detection methods are followed by edge linking techniques. After a brief discussion of the edge extraction methods, the edge combination methods are addressed.

Having already defined region boundaries by sharp changes in intensity, it

is obvious that there is an associated high spatial frequency at the edge. By high-pass filtering the image in the frequency domain, the edges of the image can be enhanced and extracted. This method is known as high-emphasis spatial frequency filtering [8]. Edge detection has also been viewed as an approximation problem. By looking at the change of intensity at an edge as a function of location, an edge can be compared to the ideal step function edge to obtain a measure of edge quality. This method is known as functional approximation [13]. The most popular method of edge extraction, however, uses gradient operators.

The gradient operator is defined as

$$\nabla f(x, y) = \frac{\partial f}{\partial x} \hat{i} + \frac{\partial f}{\partial y} \hat{j}$$

with magnitude

$$|\nabla f(x, y)| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

and direction

$$\tan^{-1} \frac{(\frac{\partial f}{\partial y})}{(\frac{\partial f}{\partial x})}$$

where $f(x, y)$ is the gray level of point (x, y) and $\hat{i}$ and $\hat{j}$ are unit vectors for the x and y axis respectively. Besides the general definition of the gradient, several operators have been developed that produce a value proportional to the change in gray level [5,14,20,22,27,28,32]. Some of the more popular operators based on 3x3 windows are the Sobel, Robert's, and Prewitt Gradients. Each of these operators can be defined as 3x3 masks as shown in Figure 2.1.

The major drawback to using edge detection methods in segmentation is that the edges extracted seldom enclose the whole region. Without good connectivity along the boundary of the object, it cannot be extracted from

the scene accurately. The methods used to link extracted edge segments are, therefore, very important in edge-based segmentation.

Robert's	Sobel	Prewitt																											
<table><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>-1</td><td>0</td><td>-1</td></tr></table>	1	0	1	0	0	0	-1	0	-1	<table><tr><td>-1</td><td>0</td><td>1</td></tr><tr><td>-2</td><td>0</td><td>2</td></tr><tr><td>-1</td><td>0</td><td>1</td></tr></table>	-1	0	1	-2	0	2	-1	0	1	<table><tr><td>-1</td><td>0</td><td>1</td></tr><tr><td>-1</td><td>0</td><td>1</td></tr><tr><td>-1</td><td>0</td><td>1</td></tr></table>	-1	0	1	-1	0	1	-1	0	1
1	0	1																											
0	0	0																											
-1	0	-1																											
-1	0	1																											
-2	0	2																											
-1	0	1																											
-1	0	1																											
-1	0	1																											
-1	0	1																											
	vertical	vertical																											
	<table><tr><td>-1</td><td>-2</td><td>-1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>2</td><td>1</td></tr></table>	-1	-2	-1	0	0	0	1	2	1	<table><tr><td>-1</td><td>-1</td><td>-1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	-1	-1	-1	0	0	0	1	1	1									
-1	-2	-1																											
0	0	0																											
1	2	1																											
-1	-1	-1																											
0	0	0																											
1	1	1																											
	horizontal	horizontal																											

Figure 2.1: Robert's, Sobel, and Prewitt Gradient Operators

Current linking methods consist of relaxation techniques, heuristic searches, dynamic programming, and curve fitting. Relaxation techniques use an iterative process to continually update the probability that a candidate edge element is really a true edge element [21,24]. In this way the requirements for classification as an edge can be relaxed when less prominent edges exist in the image. In dynamic programming, a figure of merit or cost function is calculated for different paths between the edge segments [18]. This method finds an "optimal" path between the two edge segments. Another popular method is to use line or curve fitting through edge elements [6,29]. These methods have also been used in combination with the Hough Transform to detect collinear points in order to achieve a best fit between edge segments. The consistent theme of these methods involves relaxing the requirements

for classification as an edge in areas where edge information is suppressed or nonexistent. It is very common for an image to have pieces of boundaries missing due to noise or intensity fluctuations. By relaxing edge requirements in these regions, however, a large amount of performance uncertainty is introduced. One of the purposes of introducing multi-frame techniques is to reduce this uncertainty by using only existing edge information. One of the characteristics of video imagery is that many frames will have approximately the same image but different noise. This is the basic principle behind the idea of using multiple frames to link edges.

Research carried out by the author has proven unfruitful in discovering methods of applying temporal information to link edges. Temporal information has, however, been successfully used in aiding similar aspects of image processing. It has been shown that multi-frame analysis has enhanced results of target detection from 86 to 97 % and classification from 60 to 65 % [30]. In reference to this vast improvement Helland, *et al.*, notes:

Classification was carried out in this case by analysis of edge information obtained from the single integrated image. A preferred approach would be to extract edges from all frames so as to improve the statistics of edge occurrence as well as to provide the averaging effects of the noise. [12]

The application of this approach is discussed in detail in the next chapter.

CHAPTER III

TEBS ALGORITHM

The Temporal Edge-Based Segmentor (TEBS) Algorithm is developed in this chapter. The discussion of TEBS is broken into two parts: the detection of edges and the use of temporal information. The heart of this algorithm is a simple edge detection procedure. Its uniqueness, however, lies in the use of temporal information to enhance and link edge information.

The initial application of the TEBS algorithm is the processing of forward-looking infrared (FLIR) imagery. The characteristics of FLIR image data influence the techniques used in the edge detection. The methods for implementation of temporal information, however, are independent of the data characteristics.

3.1 EDGE DETECTION

As discussed in the previous chapter, edge detection is a very powerful method of image segmentation. Not only do edges contain most of the image information, but operation on edge images also reduces the effects of intensity unpredictability and fluctuation over the image. The edge information of an intensity image is enhanced using an edge operator to obtain the gradient image. Some popular edge enhancement operators discussed in Chapter II are Robert's, Sobel, and Prewitt. The Sobel operator has been very popular in the past because of its presumably superior signal-to-noise performance. Recent studies have shown, however, that the Prewitt operator is actually better in several aspects [17].

When analyzing infrared (IR) imagery produced by low-resolution FLIR sensors, it is observed that edges tend to be smooth transitions extending over three or four pixels. The Prewitt operator maximizes the output signal-to-noise ratio (SNR) for uniform slope edges corrupted by additive Gaussian noise. It is also less sensitive than the Sobel and Robert's operators to noise of any variance [17]. Because of this optimum noise-handling capability for images containing ramped edge transitions, the Prewitt edge operator is used to extract edges from the IR image data.

The Prewitt operator is a type of symmetric operator and can be represented by the formulas

$$X(i, j) = (A2 + A3 + A4) - (A0 + A7 + A6)$$

$$Y(i, j) = (A0 + A1 + A2) - (A6 + A5 + A4)$$

with the neighborhood pixels defined as shown in Figure 3.1.

A0	A1	A2
A7	i,j	A3
A6	A5	A4

Figure 3.1: 3x3 Pixel Cell for Prewitt Gradient

The edge magnitude and phase images are created using the formulas

$$Magn(i, j) = \sqrt{X(i, j)^2 + Y(i, j)^2}$$

$$Phas(i, j) = \tan^{-1} \frac{Y(i, j)}{X(i, j)}.$$

The edge magnitude image contains a blurry outline of the object in the original image as shown in Figure 3.2. The edge phase image defines the

direction of the maximum derivative in gray shade for each pixel. This value is quantized into eight levels which simplifies the direction to one of the pixel's eight neighbors. The edge magnitude and phase can then be used to obtain a thinned edge magnitude image.

The edge magnitude image needs to be accurately thinned in order to achieve an accurate binary boundary. The thinning algorithm uses directional information acquired from the Prewitt gradient to guide the thinning process. By looking in a direction orthogonal to the blurred edge, the maximum gradient value across the edge can be saved while deleting adjacent edge values. The thinned edge image accurately represents the boundaries found by the Prewitt gradient and is shown in Figure 3.3. If the target edges are more prominent than background edges and noise, the thinned image can be globally thresholded using intensity statistics to separate object edges from clutter.

To separate object edges from edge clutter, the threshold selection is important. The threshold is calculated using the mean and standard deviation, *s.d.*, from the edge magnitude image shown in Figure 3.2. By using this statistical information, the threshold is flexible to variations in noise statistics from frame-to-frame. The threshold T is calculated as follows

$$T = mean + k \times \frac{s.d.}{1 - \frac{1}{mean}}$$

where the *mean* and *s.d.* are properties of the edge magnitude image and k is experimentally determined for the specific type of data. The value of k obviously controls the strictness of the threshold. This formula was empirically developed and is to be published in the future [1]. The threshold is applied to the entire edge image and gives the edge segments shown in Figure 3.3.

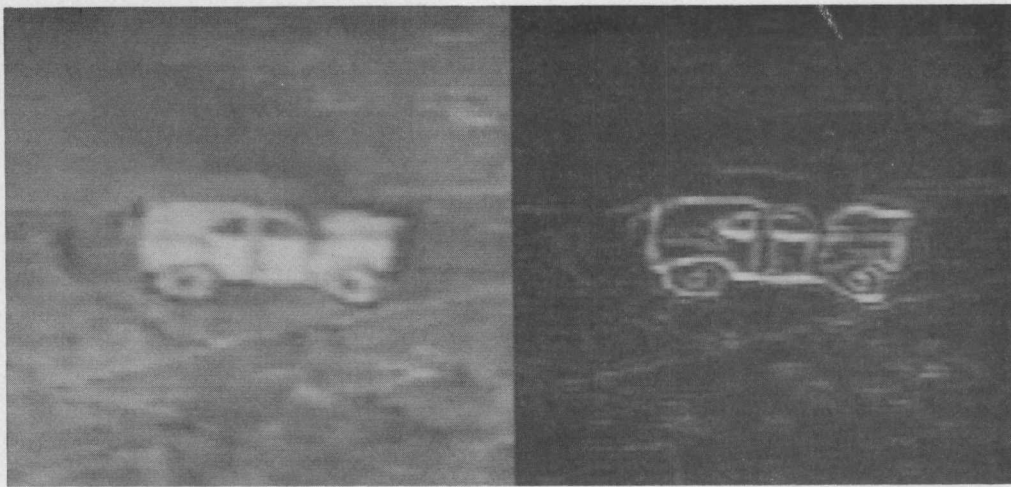


Figure 3.2: Original Image and Edge Magnitude Image

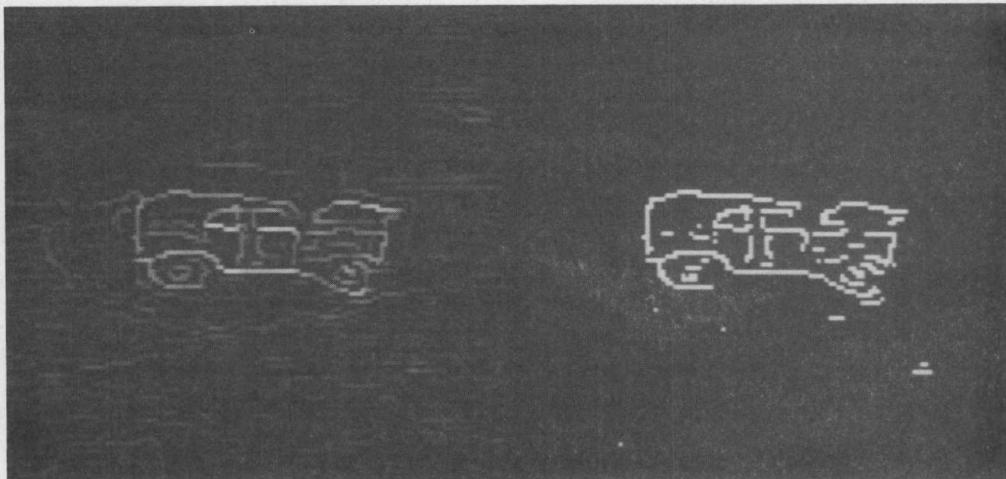


Figure 3.3: Thinned Edges and Thresholded Edges

The initial threshold is not as important in TEBS as it would be in algorithms expecting good edges immediately. The goal for threshold selection in TEBS is to keep a strict threshold that extracts edge information without any noise or background clutter. It will be shown later that temporal information in the midprocessing phase will allow relaxation of this threshold while excluding the background clutter.

3.2 TEMPORAL INFORMATION

The uniqueness of the TEBS algorithm lies in the use of temporal information in all phases of processing. The use of temporal information in preprocessing increases the SNR of the raw data. The concatenation of edge information from multiple frames is a powerful tool used to increase the SNR of edge information in midprocessing. This allows the initial edge detection techniques to be more stringent and, thus, less susceptible to noise. Edges from adjacent frames are once again used in postprocessing to link edge segments. This synergistic effort composes an algorithm which is both heuristically and mathematically appealing.

The obvious use of temporal information in preprocessing involves combination of frames to reduce the noise of the image. The procedure of image averaging is very effective in reducing the noise variance. By averaging M frames, the noise variance can be reduced by a factor of M [9]. This approach works very well when the sequence has the same signal present in each frame. This is not the case in most real applications. When the signal is changing from frame-to-frame, averaging of these frames will not only reduce the strength of the random noise, but the strength of the signal also.

A more practical approach to image averaging is accomplished using the Alpha Filter.

The Alpha Filter can be viewed as a single-pole recursive low-pass filter operating on a frame-to-frame basis [2]. It can also be described as the combining of past frames (%old) with the new frame (%new) to achieve a new averaged image. The feedback coefficient α controls the weight of the old image in the averaging scheme. A block diagram of the Alpha Filter is shown in Figure 3.4.

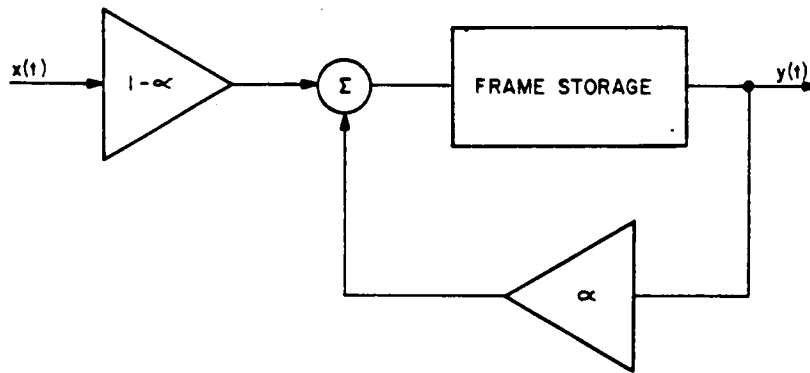


Figure 3.4: Block Diagram of Alpha Filter

The Alpha Filter can be implemented by applying the difference equation

$$y_t = (1 - \alpha)x_t + (\alpha)y_{t-1}$$

to each pixel in the image, where x_t is the current frame, y_t is the current averaged image, and $0 \leq \alpha \leq 1$. A detailed study of the Alpha Filter in the presence of noise is given in Appendix A. This study shows that the average

noise power at the output of the filter is reduced by a factor of

$$\frac{(1 - \alpha)^2}{1 - \alpha^2}$$

for the white noise case. The filter must process M sequential frames before this effect is realized, but each frame thereafter will also realize the same noise reduction. The results of simple image averaging can be obtained using the Alpha Filter by realizing that the feedback term α can be expressed in terms of M

$$\alpha = \frac{M - 1}{M + 1}.$$

This relationship is graphed in Figure 3.5.

The SNR is defined as a ratio of the contrast, C , to the variance, σ^2 .

$$SNR_{out} = \frac{C}{\sigma_{out}^2}$$

$$SNR_{in} = \frac{C}{\sigma_{in}^2}$$

where C is defined as the average difference between the object and background. It was indicated earlier that straight image averaging reduces the variance of the noise by a factor of M

$$\sigma_{out}^2 = \frac{1}{M} \times \sigma_{in}^2.$$

By substituting variance in terms of SNR and M into the equation above, the SNR becomes

$$SNR_{out} = \sqrt{M} \times SNR_{in}.$$

This relationship is graphed in Figure 3.6.

One of the problems encountered with implementation of the Alpha Filter is the choice of the feedback coefficient α . In the FLIR data used, the target in the image is "growing" due to the sensor closing on the target. Therefore,

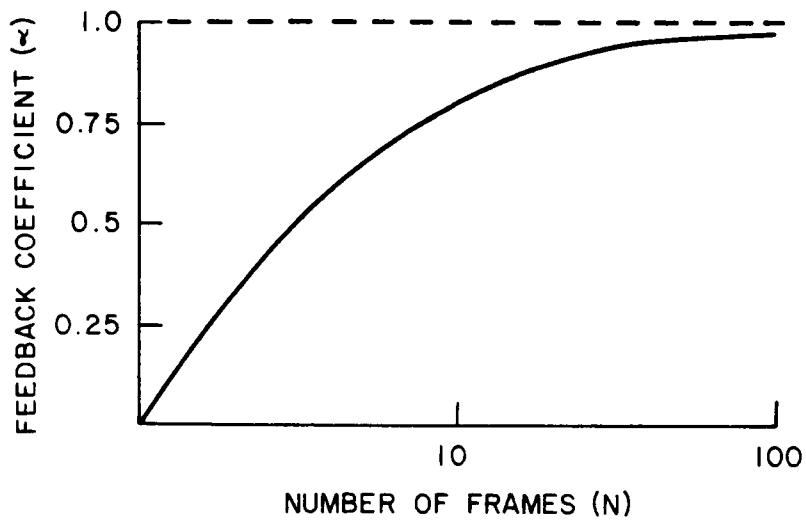


Figure 3.5: Feedback Coefficient vs. Number of Frames

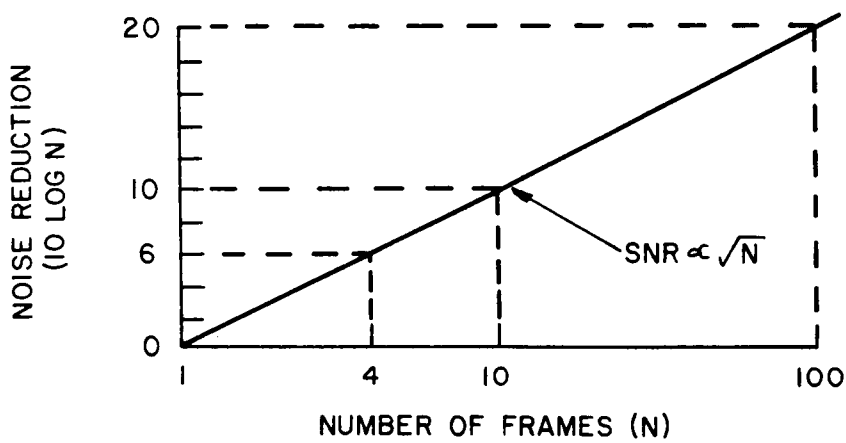


Figure 3.6: Noise Reduction vs. Number of Frames

if α is large and many frames are integrated, even though the target is registered, the edges will be blurred. The severity of this trade-off between noise reduction and edge strength depends on the data rate, sensor speed, and target motion. A value of α must be determined that reduces noise without blurring edges beyond recognition.

It is obvious that the objects must be registered from frame-to-frame in order to implement image filtering without smearing the edges of the target. This is achieved by using a simple correlation tracker that extracts a subimage from a larger scene with the target centered. Much research has been dedicated to the design of sophisticated tracking software. It is not the purpose of this thesis to design a tracker, but it is important to continually register the target. The tracker implemented uses some basic fundamentals of correlation trackers. The correlation tracker in TEBS can best be described as a fixed reference minimum difference correlation tracker. The correlation sum is calculated using the formula

$$sum(\Delta x, \Delta y) = \sum_{i=-\frac{M}{2}}^{\frac{M}{2}} \sum_{j=-\frac{N}{2}}^{\frac{N}{2}} D(i, j, \Delta x, \Delta y)$$

for an $M \times N$ subimage where

$$D(i, j, \Delta x, \Delta y) = |f(i, j) - g(x_c + i + \Delta x, y_c + j + \Delta y)|.$$

By correlating the reference image, $f(i, j)$, with the incoming scene image, $g(x_c + i + \Delta x, y_c + j + \Delta y)$, over a window $(\pm \Delta x, \pm \Delta y)$ around the previous center (x_c, y_c) , the minimum sum will tell the change needed to update the center of the object

$$x_c = x_c + \Delta x$$

$$y_c = y_c + \Delta y.$$

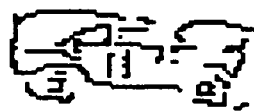
This provides the filter with the subimage that has the closest resemblance to the reference image.

The reference image is extracted at the beginning of the sequence and correlated with every incoming frame. Obviously, actual target positions can be resolved only to an integer pixel. Target motion, however, is often subpixel in nature. Each time the reference is updated, subpixel motion is not included in the actual target registration. There is a need, however, to update the reference from time to time since the target is growing and possibly moving or changing orientation with respect to the sensor. This can be done by updating the reference every N frames or by some automated process that monitors the correlation sum and updates the reference when the sum is consistently high. For the purposes of research, only 32 frames were processed at a time and reference updating is unnecessary.

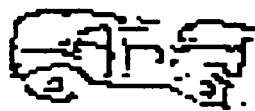
One of the most exciting applications of temporal information is the implementation in midprocessing to increase the SNR of extracted edges. In this case, the SNR is defined as

$$SNR = \frac{\text{object boundary information}}{\text{background clutter}}.$$

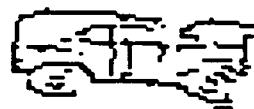
As mentioned earlier, the initial thresholding of the edges should be stringent to insure removal of noise and background clutter. To increase the SNR in the edge image, the initial threshold is relaxed in a specified region. Noise will cause the thresholded edge image from different frames to contain slightly different object boundaries. This is demonstrated in the four sequential frames shown in Figure 3.7. By summing the past N edge images together an edge image is created that has a fully connected thick boundary as shown in Figure 3.8. By lowering the threshold inside these thick edges, a significant amount of object edge information can be found. This result is shown in



SEQ01F00020



SEQ01F00021



SEQ01F00022



SEQ01F00023

Figure 3.7: Edges Extracted from Sequential Frames

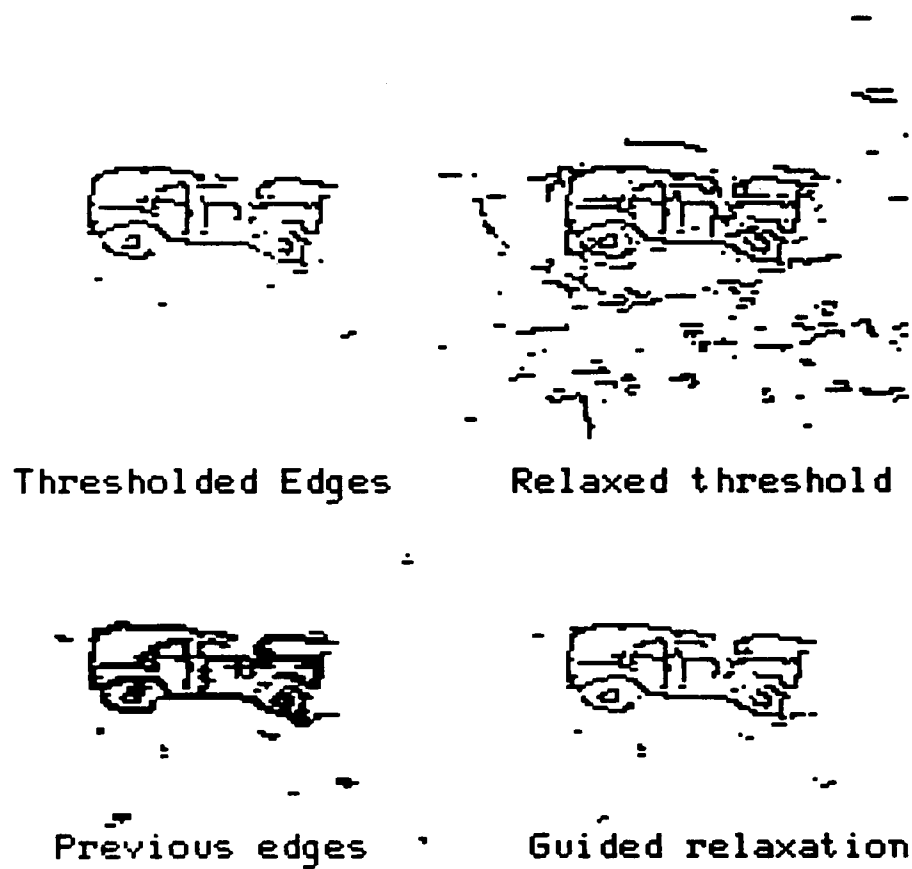


Figure 3.8: Temporal Information in Midprocessing

Figure 3.8 to contrast the amount of noise that would have been extracted at the threshold lowered without guidance. The actual procedure of guiding the relaxation is a simple process of logically ANDing the information of the thick edges and the reduced or relaxed threshold as illustrated in Figure 3.9.

The thick edges obtained in this step could be thinned and used for the boundary of the target and thus complete the boundary finding process. There is a great need in FLIR target recognition, however, to use the most current edge information to segment the target accurately. In less stringent applications, this result would be sufficient to enable target extraction. Even after the preprocessing and midprocessing techniques increase the SNR of raw data and edge information, an individual frame still has gaps in the object boundary.

The final implementation of methods using temporal information is in the postprocessing stage of edge detection. Even though gaps still exist in the object boundary, it has been demonstrated that the missing edges do exist in other frames. The basic idea behind the postprocessing technique is to preserve local connectivity of the target boundary by accumulating edges until a complete boundary is obtained. From then on, the gaps in the present frame can be filled by referring to the previous frame. This is accomplished by the algorithms CLEAN, ENDPTS, and PRESERVE found in Appendix B.

In order to keep the more complicated PRESERVE algorithm from working overtime, single-frame techniques are used on the thresholded edge image to clean and link simple imperfections in the boundary. This algorithm called CLEAN will link simple one pixel gaps in the edge image by neighbor searches, and will delete stray points. If a pixel (i, j) is not currently classified as an edge point, it will be tested to see if it should be included as edge

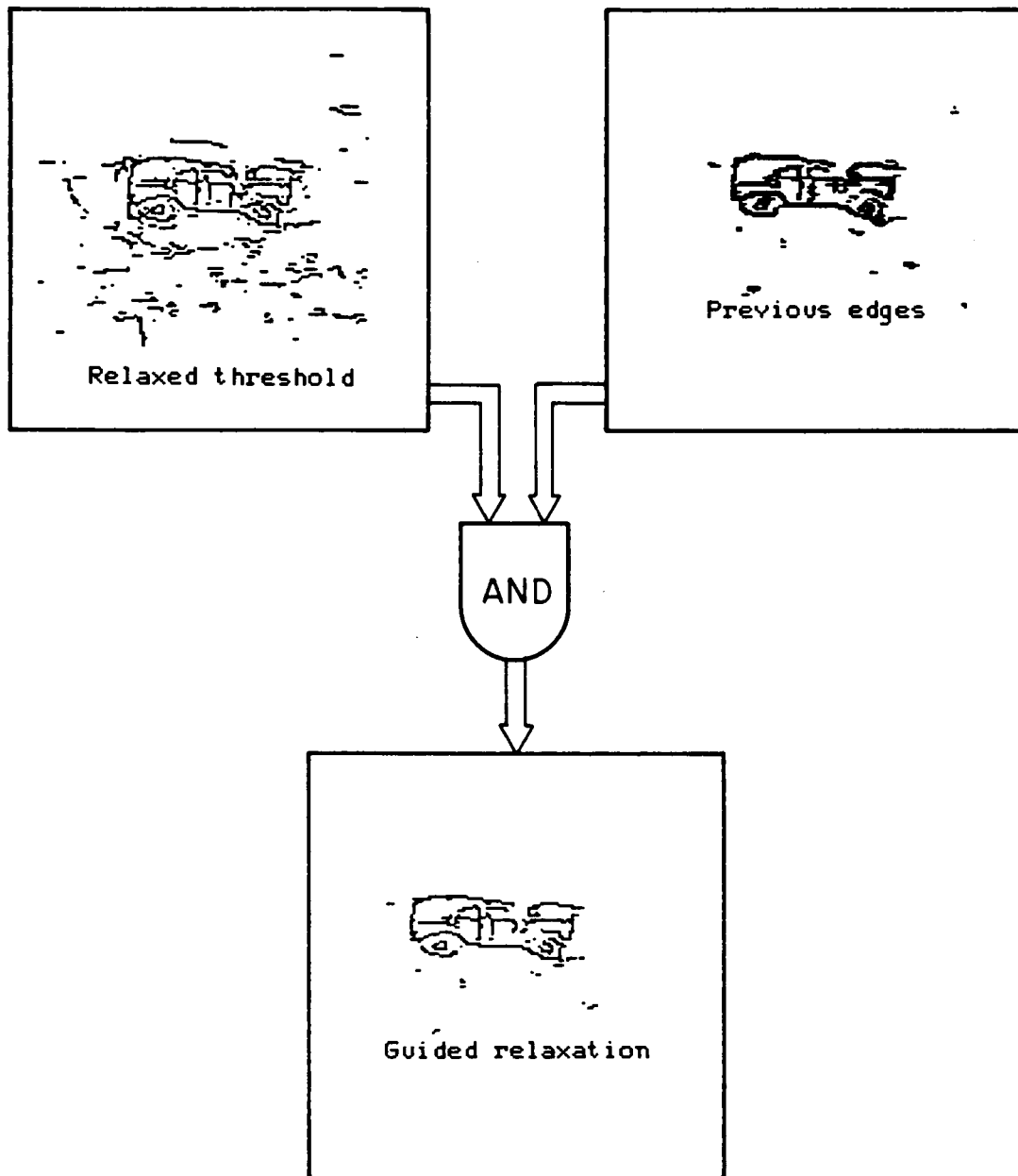


Figure 3.9: Guided Threshold Relaxation Process

information. The CLEAN subroutine will classify the pixel (i, j) shown in Figure 3.10 as an edge pixel only if there are two other edge pixels in the mask and they do not touch. The individual rules are graphically illustrated in the subroutine CLEAN in Appendix B. If the pixel (i, j) is currently classified as an edge pixel, it must pass several tests to keep its classification as proper edge information. The first test determines if (i, j) is a lone noise point and will delete it if none of the neighbors in the mask in Figure 3.10 are edge pixels. The other type of test checks to see if (i, j) is a "jetting" point. A "jetting" point is defined as a one pixel extension orthogonal to a straight edge. For example, (i, j) would be deleted if NW, N, and NE are the only other pixels in the mask in Figure 3.10 classified as edge pixels. These rules are also graphically illustrated in the CLEAN subroutine in Appendix B.

NW	N	NE
W	i, j	E
SW	S	SE

Figure 3.10: Directional Mask for Single-Frame Postprocessing

Once the boundary is cleaned, the endpoints can be located to denote gaps in the boundary. This is achieved by again using the directional mask shown in Figure 3.10. When an edge point (i, j) is found and five points covering three sides under the mask are not edge points, that edge pixel is flagged as an endpoint. In other words, (i, j) is an endpoint if any one of the following equations is true

$$W + NW + N + NE + E = 0$$

$$N + NW + W + SW + S = 0$$

$$N + NE + E + SE + S = 0$$

$$W + SW + S + SE + E = 0$$

where the directions are defined as binary points (*edge pixel* = 1) under the mask shown in Figure 3.10. These rules are also graphically illustrated in the ENDPTS subroutine in Appendix B.

The endpoints located are used to guide the PRESERVE algorithm in concatenating edge information to fill the gaps in the boundary. The final resulting boundary in the previous frame may contain the information missing in the present frame. Since edge boundaries in adjacent frames may be shaped differently or may not overlap, as demonstrated in Figure 3.7 on page 19, simple addition of local edge information is not adequate. It is necessary to correlate around the vicinity of the missing edge to align local edges with the previous frame. This is done in a manner similar to the CORR routine discussed previously. Simple difference correlation cannot be used since the edge image is binary. By weighting the occurrence of overlapping and adjacent edges between frames, a correlation figure can be calculated. When correlation is performed and edge pixels are found to overlap the correlation sum is updated by

$$sum(\Delta x, \Delta y) = sum(\Delta x, \Delta y) + c$$

and when edge pixels are adjacent between frames

$$sum(\Delta x, \Delta y) = sum(\Delta x, \Delta y) + k$$

where $c > k > 0$. The maximum correlation sum denotes the best overlapping of edge information, and the local edge information is then summed into

the present frame. An example of how this method preserves the connectivity of the edges in real FLIR data is shown in Figure 3.11.

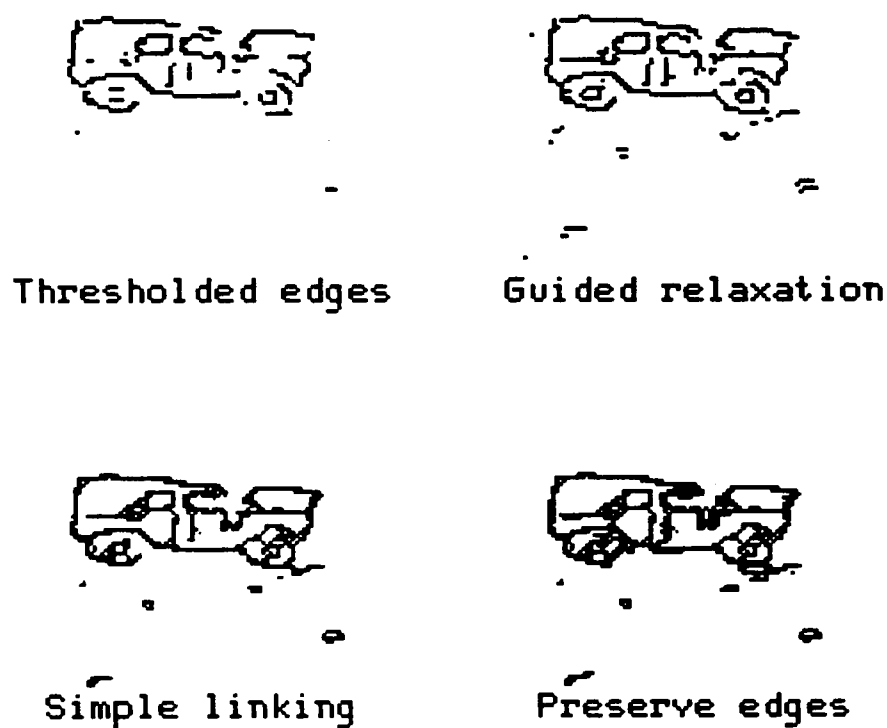


Figure 3.11: Temporal Information in Postprocessing

CHAPTER IV

RESULTS

The TEBS algorithm described in the previous chapter was implemented in FORTRAN on a VAX 11/780 computer. The main program and subroutines are listed in Appendix B. The algorithm was applied to both synthetic and real image data. The intermediate results on synthetic data are shown to reiterate the ideas used to arrive at the final goal of segmentation. The explanation of the real data results is on a more technical level, and the algorithms used to accomplish intermediate results are discussed.

4.1 SYNTHETIC IMAGES

The synthetic images are composed of sequences of 32 frames with object movement of 0.4 pixel per frame and growth of 0.1 pixel per frame. This subpixel movement and growth is characteristic of real FLIR image data due to the sensor closing on the target. The object is a simple 10 pixel square and has an intensity of 158 of 256 while the background is 98. This gives a contrast of 60 while centering the mean at an intensity of 128 so noise response can be easily tested. Each frame of this sequence of moving and growing squares is then corrupted with additive uncorrelated Gaussian noise with zero mean. The noise is added to create SNRs of four, two, and one. The TEBS algorithm was applied to each of these sequences to determine its noise handling capability.

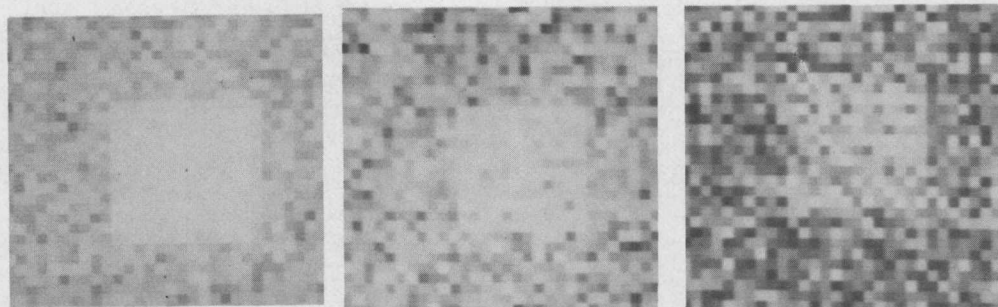
The first processing in TEBS involves object registration and alpha filtering to improve the SNR of the raw data. The choice of the feedback

coefficient α is very important for different SNRs. The value chosen was guided by considering the SNR of the image and the sharpness of the edges in the filtered image. The values of alpha were experimentally chosen to be $\alpha = 0.9$ ($SNR = 1$), $\alpha = 0.7$ ($SNR = 2$), and $\alpha = 0.5$ ($SNR = 4$). The results in Figure 4.1 show the ability of the Alpha Filter to reduce the noise in the original image.

The results of the edge detection process isolated to a single frame are shown in Figure 4.1. This result demonstrates the need for edge enhancement and linking procedures. This would be the type of edge image presented to the edge combination techniques if temporal information was not used. By using temporal information, however, edge enhancement is provided during the edge detection procedure. The results of guiding the threshold relaxation are also shown in Figure 4.1. The relaxed edges are cleaned by linking simple gaps and deleting stray points as shown in Figure 4.2. By preserving the edge information existing in the previous frame, remaining gaps are filled as shown in Figure 4.2. Once a boundary with good connectivity is established, the object can be segmented from the original image as shown in Figure 4.2.

4.2 FLIR DATA

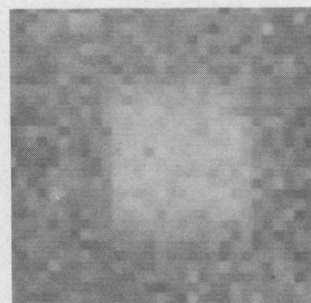
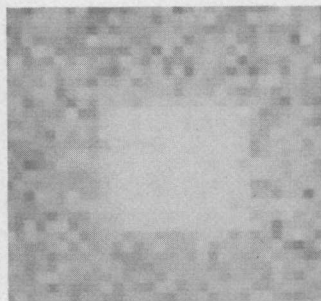
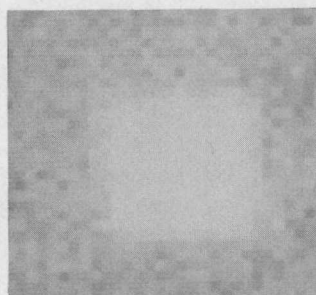
The real data heavily influenced the development of the TEBS algorithm, and yet the methods implemented remain general enough to consider applications to other video imagery. The data consisted of sequences of FLIR images digitized at a 30 Hz. frame rate. The scene consisted of various types of background clutter and a "large" target. This type of target is close enough to the FLIR sensor that the image contains distinguishing details and automatic recognition can be attempted. This is in contrast to the better



$SNR = 4$

$SNR = 2$

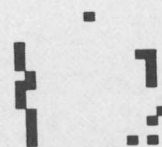
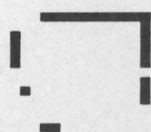
$SNR = 1$



$\alpha = 0.5$

$\alpha = 0.7$

$\alpha = 0.9$



Thresholded Edges



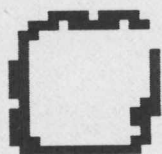
Relaxed Edges

Figure 4.1: Filtering and Edges Extraction with Synthetic Data

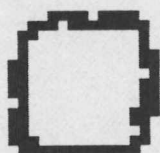
$SNR = 4$

$SNR = 2$

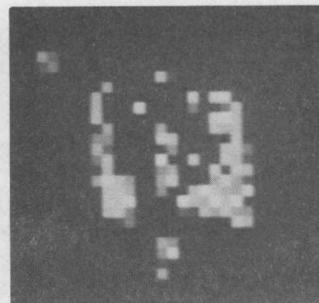
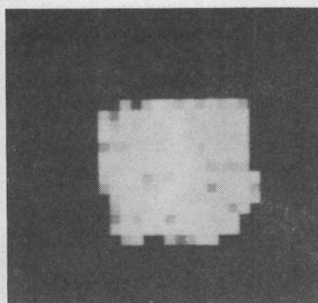
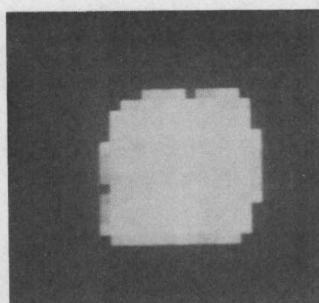
$SNR = 1$



Simple Linking



Preserving Edges



Target Segmented

Figure 4.2: Edge Linking and Object Segmentation with Synthetic Data

known FLIR imagery of bright "blobs" in a scene. The size of the scene images is 512x512 pixels, and the targets range from 64x64 to 128x128 pixels. The size affects only the speed of the TEBS algorithm. Several types of motion are present in this type of imagery. It is important to realize that the FLIR sensor is mounted on some type of aircraft and is automatically tracking a target as it closes in. Therefore, the first type of motion present is the motion of the sensor. This motion causes the target to grow in the fixed window of the sensor's resolution. As would be expected, a target is unlikely to sit still during the approach of the aircraft, so the target can be expected to move also. This can cause a change in orientation and thus a change in the shape of the target in the scene. The movement and growth of the target would make it seem impractical to apply the temporal information techniques discussed earlier. It is important to remember that the 30 Hz. frame acquisition causes adjacent frames to be very similar with respect to object size and location. The growth and motion of the target can be described as subpixel and thus indistinguishable between adjacent frames. Precautions against averaging too many frames have already been discussed. The data acquisition rate, along with the speed of closure on the target, are important parameters to consider when deciding on the number of frames to combine. The results of the real data are presented in a way that will lead the reader through the sequence of algorithms used and how their intermediate results contribute to the ultimate goal of segmentation.

The sequence of Figures 4.3 to 4.10 demonstrates the chronological order in which the TEBS algorithm uses intermediate results to arrive at the final goal of segmentation. The original scene is shown in Figure 4.3.

The target is extracted initially by providing the EXTRACT subroutine with the initial target center and size. The EXTRACT routine simply

CRANESEST CREST



Figure 4.3: Original Scene Image

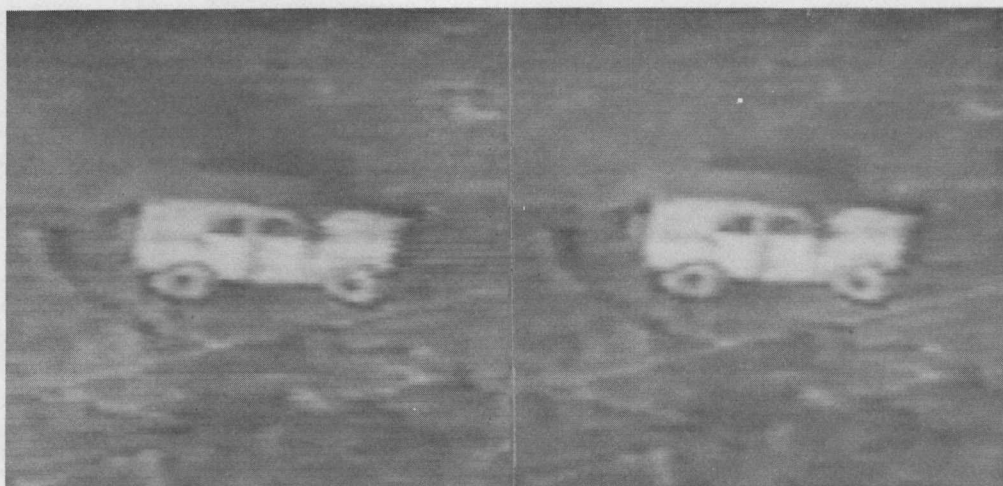


Figure 4.4: Extracted Subimage and Filtered Subimage

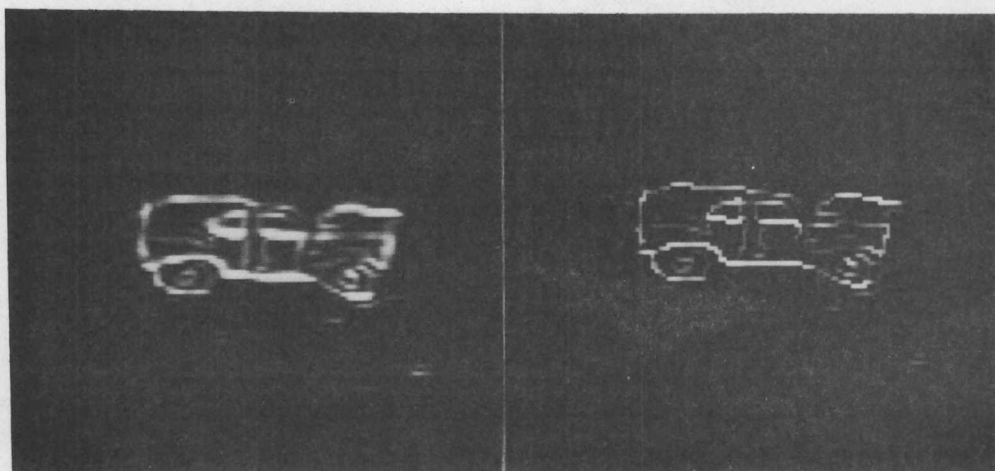


Figure 4.5: Edge Magnitude Image and Thinned Edge Image



Figure 4.6: Thresholded Edges

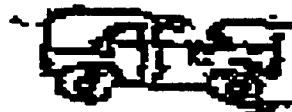


Figure 4.7: Thick Edges



Figure 4.8: Relaxed Edges



Figure 4.9: Cleaned Edges and Preserved Edges

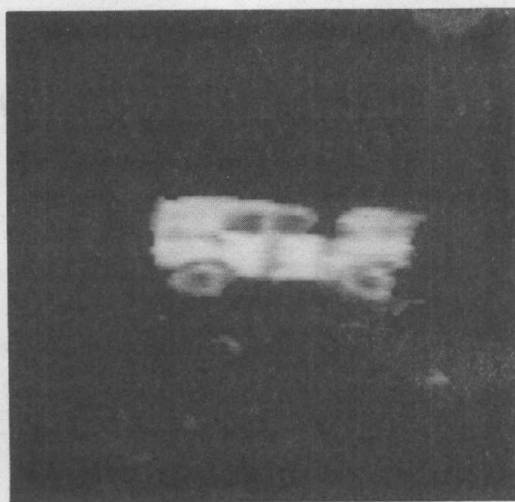


Figure 4.10: Segmented Object

removes a subimage of a given size around a center coordinate. This subimage is used as a reference to register sequential frames by the subroutine CORR. The routine CORR performs a correlation on the previous center ± 2 pixels around the center to account for movement. The CORR routine then extracts the subimage from the incoming frame around the center having the best correlation sum. The registered subimages are then filtered by the ALPHA routine. The present averaged image is updated by the new subimage in ALPHA for each new frame. The results of these three steps are shown in Figure 4.4. The edges of the filtered image are enhanced by the PREWIT subroutine. This subroutine finds the edge magnitude and direction for each pixel in the subimage. The resulting blurry edges can be thinned by processing the edge magnitude and direction images by the THINEDGES subroutine. The edge magnitude image is thinned by deleting pixels in a direction orthogonal to the direction of the maximum gradient on the thick edge. These results are shown in Figure 4.5. The boundary of the object is thresholded from the thinned edge image by applying the threshold in the APSTHRES subroutine. The threshold applied extracts little noise but leaves gaps in the object boundary as demonstrated in Figure 4.6. The past ten thresholded edge images can be summed to obtain a thick, connected boundary as shown in Figure 4.7. These thick edges are used to guide relaxation of the threshold and result in better edge content on the object boundary without noise added. This result is shown in Figure 4.8. The simple gaps and noise that still exist are processed by the routine CLEAN before the subroutine PRESERVE is called. The PRESERVE subroutine links the remaining gaps using edge information accumulated in the previous frame. The results of CLEAN and PRESERVE process are shown in Figure 4.9. The final step is to extract the object using the completed boundary, thus

performing the final goal of segmentation. This is implemented by a simple series of routines FILLIN and AND. The FILLIN subroutine checks for edge information around the pixel in question to determine if it is really inside the boundary. A pixel can be classified as inside the boundary if edge information is found in all eight directions around it. This routine makes small errors due to inaccurate classification of background pixels in areas of complex boundary shapes. The AND subroutine simply uses the result of the FILLIN routine to extract the object from the original image as shown in Figure 4.10. The success of the algorithm along with recommendations for further research are discussed in the following and final chapter.

CHAPTER V

CONCLUSIONS AND RECOMMENDATIONS

As shown in the previous chapter, the use of temporal information is a very powerful aid in enhancing an edge-based segmentor. The success demonstrated should not come as a surprise. As discussed in Chapter II, the approach of concatenating existing edge information from surrounding frames is intuitively more appealing than single frame approaches. Most edge detection methods will find different boundaries and leave different gaps for each frame in a sequence of images. Any of these methods can, therefore, be enhanced by the use of temporal information as described. Further research is needed to find more ways in which temporal information can be used to enhance edge-based segmentors. Some suggestions are to combine the thinned edge information by alpha filtering techniques before thresholding. Several places in the edge detection scheme will allow midprocessing use of temporal information. Methods could be developed that use temporal information in other phases of processing to aid in boundary extraction.

Suggested improvements to the present algorithm require refinement of the ideas implemented here. A more sophisticated and accurate tracker will insure more exact registration and thus prevent any edge blurring due to motion. At the postprocessing phase, the PRESERVE algorithm is very important. Images with very complex boundaries can cause difficulties. The method presented fills gaps by summing edges from the previous frame. This should be made more flexible to insure proper connection and also discard clutter. This can be done a number of ways, one of which would be to thin the edges accurately if PRESERVE introduces any clutter. Another method

would be to check the local connectivity in the previous frame and use this local information in a flexible way to freely guide the linking of gaps in the current frame.

In further research and development, automation could be introduced in a number of ways. As previously mentioned, the correlation sum could be used to decide when to update the correlation reference. This sum can also be monitored to regulate the value of α from frame-to-frame. When the correlation sum is higher than expected, the current frame is significantly different than the reference. If this type of change is not noise, the value of α should be increased to weigh this frame heavily in the averaging scheme. With further refinement of the algorithms and further implementations of automation, the research provided in this thesis can be the basis for a very successful edge-based segmentor.

BIBLIOGRAPHY

BIBLIOGRAPHY

- [1] Ploysongsang, A., Conversation Concerning Threshold Formula to be Published, August 20, 1985.
- [2] Brown, R.G. (1963) *Smoothing, Forecasting and Prediction of Discrete Time Series*, Prentice Hall, pp. 143.
- [3] Carlton, S.G. and Mitchell, O.R. (1977), "Image Segmentation using Texture and Gray Level," *IEEE Conf. Pattern Recognition and Image Processing*, Troy, New York, pp. 387-391.
- [4] Coleman, G.B. (1977), "Image Segmentation by Clustering," *Report 750*, University of Southern California Image Processing Institute.
- [5] Duda, R.O. and Hart, P.E. (1973), *Pattern Classification and Scene Analysis*, Wiley, New York.
- [6] Duda, R.O. and Hart, P.E. (1972), "Use of the Hough Transform to Detect Lines and Curves in Pictures," *Communs ACM*, vol. 15, pp. 11-15.
- [7] Feldman, J.A. and Yakimovsky, Y. (1974), "Decision Theory and Artificial Intelligence, I. A Semantics Based Region Analyzer," *Artificial Intelligence*, vol. 5, pp. 349-371.
- [8] Fu, K.S. and Mui, J.K. (1981), "A Survey on Image Segmentation," *Pattern Recognition*, vol. 13, pp. 3-16.
- [9] Gonzalez, R.C. and Wintz, P. (1978), *Digital Image Processing*, New York: Wiley and Sons.
- [10] Gupta, J.N., and Wintz, P.A. (1974), "Computer Processing Algorithm for Locating Boundaries in Digital Pictures," *Proc. Int. Joint Conf. Pattern Recognition*, pp. 155-156.
- [11] Gupta, J.N. and Wintz, P.A. (1974), "Multi-image Modeling," *Technical Report TR-EE 74-24*, School of Electrical Engineering, Purdue University.
- [12] Helland, A.R., Willett, T.J. and Tisdale, G.E. (1981), "Application of Image Understanding to Automatic Tactical Target Acquisition," *SPIE Proceedings*, vol. 281, pp. 26-31.
- [13] Hueckel, M. (1971), "An Operator which Locates Edges in Digital Pictures," *J. Ass. Comput. Mach.*, vol. 18, pp. 113-125.

- [14] Kirsch, R. (1971) "Computer Determination of the Constituent Structure of Biological Images," *Computer Biomed. Res.*, vol. 4, pp. 315-328.
- [15] Klinger A. (1973) "Data Structures and Pattern Recognition," *Proc. 1st Int. Joint Conf. Pattern Recognition*, pp. 497-498.
- [16] Klinger, A. and Dyer, C.R. (1975) "Experiments in Picture Representation using Regular Decomposition," *Computer Graphics and Image Processing*, vol. 4, pp. 68-105.
- [17] Nasburg, R.E. and Lineberry, M. (1981), "Noise Effects for Edge Operators," *SPIE Proceedings*, vol. 292, pp. 277-287.
- [18] Montanari, U. (1971), "On the Optimal Detection of Curves in Noisy Pictures," *Communs ACM*, vol. 14, pp. 335-345.
- [19] Muerle, J.L. and Allen, D.C. (1968), "Experimental Evaluation of Techniques for Automatic Segmentation of Objects in a Complex Scene," *Pictorial Pattern Recognition*, pp. 3-13.
- [20] Prewitt, J.M.S. (1970), "Object Enhancement and Extraction," *Picture Processing and Psychopictorics*, pp. 75-149.
- [21] Riseman, E.M. and Arbib, M.A. (1977), "Computational Techniques in the Visual Segmentation of Static Scenes," *Computer Graphics and Image Processing*, vol. 6, pp. 221-276.
- [22] Roberts, L.G. (1965), "Machine Perception of Three Dimensional Solids," *Optical and Electro-Optical Information Processing*, pp. 159-197.
- [23] Robertson, T.V., Fu, K.S. and Swain, P.H. (1973), "Multispectral Image Partitioning," *Technical Report TR-EE73-26*, School of Electrical Engineering, Purdue University.
- [24] Rosenfeld, A. (1977), "Iterative Methods in Image Analysis," *IEEE Conf. Pattern Recognition and Image Processing*, Troy, New York, pp. 14-20.
- [25] Rosenfeld, A., Hummel, R. and Zucker, S. W. (1976), "Scene Labeling by Relaxation Operations," *IEEE Trans. Systems, Man Cybernet.*, SMC-6, pp. 420-433.
- [26] Rosenfeld, A. and Kak, A.C. (1976), *Digital Picture Processing*, Academic Press, New York.

- [27] Rosenfeld, A. and Thurston, M. (1971), "Edge and Curve Detection for Visual Scene Analysis," *IEEE Trans. Comput.*, C-20, pp. 562-569.
- [28] Rosenfeld, A., Thurston, M. and Lee, Y. (1972), "Edge and Curve Detection Further Experiments," *IEEE Trans. Comput.*, C-21, pp. 677-715.
- [29] Shapiro, S.D. (1975), "Transformations for the Computer Detection of Curves in Noisy Pictures," *Computer Graphics and Image Processing*, vol. 4, pp. 328-338.
- [30] "Study of Automatic Tracking of Machine-Cued Targets," Final Report to U.S. Army, Frankford Arsenal, on Contract DAAA25-72-C-0154, April 15, 1974.
- [31] Tsuji, S. and Fujiwara, J. (1974), "Linguistic Segmentation of Scenes into Regions," *Proc. 2nd Int. Joint Conf. Pattern Recognition*, pp. 104-108.
- [32] Wechsler, H. and Kikode, M. (1977), "A New Edge Detection Technique and its Implementation," *IEEE Trans Systems, Man Cybernet.*, SMC-7, pp. 827-836.
- [33] Weszka, Joan S. (1978), "SURVEY - A Survey of Thresholding Selection Techniques," *Computer Graphics and Image Processing*, vol. 7, pp. 259-265.
- [34] Yakimovsky, Y. and Feldman, J. (1973), "A Semantics-Based Decision Theory Region Analyzer," *Proc. Int. Joint Conf. Artificial Intelligence*, Stanford, California, pp. 580-588.
- [35] Yoo, J.R. and Huang, T.S. (1978), "Image Segmentation by Unsupervised Clustering and its Applications," *TR-EE 78-19*, Purdue University.

APPENDICES

APPENDIX A

ALPHA FILTER NOISE REDUCTION

The transfer function of the Alpha Filter feedback system shown in Figure 3.4 on page 14 is given by

$$H(Z) = \frac{Y(Z)}{X(Z)} = Z^{-1} \frac{1 - \alpha}{1 - \alpha Z^{-1}}.$$

The output noise of the system has a power spectral density of

$$P_{YY}(\omega) = |H(e^{j\omega})|^2 P_{XX}(\omega)$$

where $P_{XX}(\omega)$ is the power spectral density of the input.

If the input is white noise, $P_{XX}(\omega) = \sigma_x^2$, the total average noise output is

$$\phi_{YY}(0) = \frac{1}{2\pi} \int_{-\pi}^{\pi} |H(e^{j\omega})|^2 \sigma_x^2 d\omega$$

where $\phi_{YY}(\omega)$ is the autocorrelation function of the output.

By substituting in the alpha filter transfer function, the result is

$$\phi_{YY}(0) = \frac{(1 - \alpha)^2}{1 - \alpha^2} \sigma_x^2.$$

Thus, the alpha filter reduces the average noise power by a factor of

$$\frac{(1 - \alpha)^2}{1 - \alpha^2}.$$

APPENDIX B

TEBS PROGRAM AND SUBROUTINES

The following listings were written in FORTRAN on a VAX 11/780 running VMS 4.1. VAX FORTRAN is a superset of FORTRAN-77. The first program is called TEBS and is a driver routine for the subroutine that follow. All of the code is well documented and able to further explain the implementation of the methods discussed in the body of this thesis. The subroutines VIMALL, VIMZER, VIMREL, and VIMPUT are not listed but their function is well documented in the main program.

```

C
C*****
C
C
C    Temporal Edge Based Segmentation Program
C
C
C
C
C    Purpose:      To find the edges of a target and segment it from
C                  the surrounding background using an edgebased
C                  segmentation algorithm enhanced by the use of
C                  temporal information.
C
C
C    Process:      o register target image by fixed reference correlation
C
C                  o remove a subimage with the target centered
C
C                  o alpha filter to suppress noise
C
C                  o find edges using Prewitt Gradient
C
C                  o thin and threshold these edges
C
C                  o sum a window of these edges to form thick edges
C
C                  o fill holes in current frame edges
C
C                  o continue in loop until sequence is over
C
C
C
C
C    Inputs:       alfa          alpha filter feedback factor
C                  ccol         center column of target in scene image
C                  crow         center row of target in scene image
C                  fact         threshold factor
C                  mask         size of gradient mask
C                  nmbr         number of frames to sum for 'thic'
C                  seq1         sequence length
C                  seqn         sequence number
C                  sfrm         starting frame
C                  wdsz         size of target subimages
C
C
C
C
C    Outputs:      avrg          alpha filter of original subimages
C                  clen         binary edges cleaned of small defects
C                  conn         final edges with gaps linked
C                  magn         edge magnitude image
C                  nend         edges with endpoints marked
C                  phas         edge phase image
C                  rlax         relaxed thresholded edges
C                  segm         segmented image
C                  thic         'thrs' combined for last 10 frames
C                  thrs         thinned and thresholded edges
C
C*****
C

```

```

c      Program TEBS
c
c      -- declare all variables --
c
Implicit      none

Integer*2     trs2
Integer*2     lastpos
Integer*2     crow , ccol
Integer*2     seqn , seql
Integer*2     sfrm , fram , efrm
Integer*2     hsiz , vsiz , wdsz
Integer*2     fact , mask , nmbr , lbnd

Integer*4     byt2 , stat
Integer*4     hdr1 , hdr2 , hdr3 , hdr4
Integer*4     hdr5 , hdr6 , hdr7 , hdr8
Integer*4     hdr9 , hdr10 , hdr11 , hdr12
Integer*4     hdr13 , hdr14 , hdr15 , hdr16
Integer*4     hdr17 , hdr18
Integer*4     img1 , img2 , subl , sub2
Integer*4     refr , magn , phas , avrg
Integer*4     thin , thrs , thic , rlax
Integer*4     nend , clen , conn , temp
Integer*4     fill , segm

Real*4        alfa

Character*32   file           ! current VIM image to be output
Character*32   pfil           ! image of past frame
Character*32   cfil           ! image of current frame

c
c      -- inputs --
c

type * , ' Sequence number to process = ?? '
accept 10 , seqn
10  format ( 1i2.2 )

type * , ' First frame in sequence = ?? '
accept 20 , sfrm
20  format ( 1i5.5 )

type * , ' Number of frames in sequence = ?? '
accept * , seql

type * , ' Target center row = ?? '
accept * , crow

type * , ' Target center column = ?? '
accept * , ccol

type * , ' Target window size = ?? '
accept * , wdsz

type * , ' Alpha ( feedback coeff. ) = ?? '
accept * , alfa

```

```

type * , seqn, sfrm, seq1, crow, ccol, wdsz, alfa

C
C
C      -- parameters --

mask = 3          ! 3x3 masks
fact = 2          ! threshold factor parameter
nmbr = 10         ! number of frames to sum for 'thic'
lbnd = 255 - nmbr ! lower bound for objects. in 'thic'

C
C      -- concatenate info. received to get filenames in sequence --
C

fram = sfrm + 1    ! increment frame count to get 2nd image

encode(16,100,pfil) 'seq', seqn, 'f', sfrm, '.img'      ! 1st image
100 encode(16,100,cfil) 'seq', seqn, 'f', fram, '.img'  ! 2nd image
    format ( la4 , li2.2 , la1 , li5.5 , la4 )

type * , ' Begin sequence at ==> ' , cfil

C
C      -- allocate space for past and present images --
C

Call VIMALL ( pfil, hdr1, img1, vsiz, hsiz, 1, 'mod', stat )
If ( .not. stat ) Call EXIT ( stat )

Call VIMALL ( cfil, hdr2, img2, vsiz, hsiz, 1, 'mod', stat )
If ( .not. stat ) Call EXIT ( stat )

C
C      -- allocate space for resulting images --
C

Call VIMALL ( ' ' , hdr3, refr, wdsz, wdsz, 1, 'new', stat )
If ( .not. stat ) Call EXIT ( stat )

Call VIMALL ( ' ' , hdr4, sub1, wdsz, wdsz, 1, 'new', stat )
If ( .not. stat ) Call EXIT ( stat )

Call VIMALL ( ' ' , hdr5, sub2, wdsz, wdsz, 1, 'new', stat )
If ( .not. stat ) Call EXIT ( stat )

Call VIMALL ( ' ' , hdr6, avrg, wdsz, wdsz, 1, 'new', stat )
If ( .not. stat ) Call EXIT ( stat )

Call VIMALL ( ' ' , hdr7, magn, wdsz, wdsz, 1, 'new', stat )
If ( .not. stat ) Call EXIT ( stat )

Call VIMALL ( ' ' , hdr8, phas, wdsz, wdsz, 1, 'new', stat )
If ( .not. stat ) Call EXIT ( stat )

Call VIMALL ( ' ' , hdr9, thin, wdsz, wdsz, 1, 'new', stat )
If ( .not. stat ) Call EXIT ( stat )

Call VIMALL ( ' ' , hdr10, thrs, wdsz, wdsz, 1, 'new', stat )
If ( .not. stat ) Call EXIT ( stat )

```

```

Call VIMALL ( ' ', hdr11, relax, wdsz, wdsz, 1, 'new', stat )
If ( .not. stat ) Call EXIT ( stat )

Call VIMALL ( ' ', hdr12, clen, wdsz, wdsz, 1, 'new', stat )
If ( .not. stat ) Call EXIT ( stat )

Call VIMALL ( ' ', hdr13, thic, wdsz, wdsz, 1, 'new', stat )
If ( .not. stat ) Call EXIT ( stat )

Call VIMALL ( ' ', hdr14, nend, wdsz, wdsz, 1, 'new', stat )
If ( .not. stat ) Call EXIT ( stat )

Call VIMALL ( ' ', hdr15, conn, wdsz, wdsz, 1, 'new', stat )
If ( .not. stat ) Call EXIT ( stat )

Call VIMALL ( ' ', hdr16, temp, wdsz, wdsz, 1, 'new', stat )
If ( .not. stat ) Call EXIT ( stat )

Call VIMALL ( ' ', hdr17, fill, wdsz, wdsz, 1, 'new', stat )
If ( .not. stat ) Call EXIT ( stat )

Call VIMALL ( ' ', hdr18, segm, wdsz, wdsz, 1, 'new', stat )
If ( .not. stat ) Call EXIT ( stat )

type *, ' -- Virtual Memory Allocated -- '

```

c
c
c

```

-- initialize image memory --

```

```

byt2 = 2. * wdsz * wdsz          ! no. bytes of I*2 image

Call VIMZER ( %Val ( refr ), byt2 )
Call VIMZER ( %Val ( sub1 ), byt2 )
Call VIMZER ( %Val ( sub2 ), byt2 )
Call VIMZER ( %Val ( avrg ), byt2 )
Call VIMZER ( %Val ( magn ), byt2 )
Call VIMZER ( %Val ( phas ), byt2 )
Call VIMZER ( %Val ( thin ), byt2 )
Call VIMZER ( %Val ( thrs ), byt2 )
Call VIMZER ( %Val ( relax ), byt2 )
Call VIMZER ( %Val ( clen ), byt2 )
Call VIMZER ( %Val ( thic ), byt2 )
Call VIMZER ( %Val ( nend ), byt2 )
Call VIMZER ( %Val ( conn ), byt2 )
Call VIMZER ( %Val ( temp ), byt2 )
Call VIMZER ( %Val ( fill ), byt2 )
Call VIMZER ( %Val ( segm ), byt2 )

```

```

type *, ' -- Virtual Memory Initialized -- '

```

c
c
c

```

-- extract subimage of target from scene for correlation reference --

```

```

type *, ' ** Call EXTRACT ** '

```

```

Call EXTRACT ( %Val(img1),          ! large scene image
               hsiz, vsiz,          ! size of large scene image

```

```

.           wdsz,                ! size of subimage
.           crow, ccol,          ! target location in large image
.           %Val(refr) )         ! target subimage reference

C
C       -- initialize 'sub1' with correlation reference subimage --
C

type * , ' ** Call SAVE ** '

Call SAVE ( %Val(refr),          ! correlation reference
.           wdsz,                ! size of subimages
.           %Val(sub1) )         ! past target subimage

C
C       -- initialize bounds for frames processed in loop --
C

fram = fram + 1                ! increment frame count
efrm = sfrm + seq1             ! end frame is first frame + number of frames

C
C       -- begin loop for processing sequence of frames --
C

do fram = fram , efrm

C
C       ** correlate scene image -- center target in subimage **
C

type * , ' ** Call CORR ** '

Call CORR ( %Val(refr),          ! correlation reference subimage
.           %Val(img2),          ! next frame large scene image
.           hsiz, vsiz,          ! size of large scene image
.           wdsz,                ! size of target subimage
.           crow, ccol,          ! location of target in scene image
.           %Val(sub2) )         ! target subimage removed

file = cfil(1:lastpos(cfil)-4)//'.sub' ! write output of
Call VIMPUT ( %Val (sub2) , file , 1 , stat ) ! subroutine to
type * , ' File created ==> ' , file         ! disk file

C
C       ** alpha filter target subimages **
C

type * , ' ** Call ALPHA ** '

Call ALPHA ( %Val(sub1),         ! previous filtered subimage
.           %Val(sub2),         ! new target subimage
.           wdsz,               ! size of subimages
.           alfa,               ! filter feedback coeff.
.           %Val(avrg) )        ! alpha filter of input images

file = cfil(1:lastpos(cfil)-4)//'.avg' ! write output of
Call VIMPUT ( %Val (avrg) , file , 1 , stat ) ! subroutine to
type * , ' File created ==> ' , file         ! disk file

```

```

C
C      -- save 'avrg' for alpha filter update next round --
C

type * , ' ** Call SAVE ** '

Call SAVE ( %Val(avrg),
            wdsz,
            %Val(sub1) )

C
C      -- release memory allocated in loop --
C

Call VIMREL ( %Val(img2), stat )
If ( .not. stat ) Call EXIT ( stat )

C
C      ** find edges of target **
C

type * , ' ** Call PREWIT ** '

Call PREWIT ( %Val(avrg),      ! target subimage
              wdsz,           ! size of subimages
              mask,           ! size of gradient mask
              %Val(magn),      ! magnitude of edge gradient
              %Val(phas) )    ! phase of edge gradient

file = cfil(1:lastpos(cfil)-4)//'.mgn'
Call VIMPUT ( %Val(magn) , file , 1 , stat )
type * , ' File created ==> ' , file

file = cfil(1:lastpos(cfil)-4)//'.pha'
Call VIMPUT ( %Val(phas) , file , 1 , stat )
type * , ' File created ==> ' , file

C
C      -- initialize 'thin' with 'magn' --
C

type * , ' ** Call SAVE ** '

Call SAVE ( %Val(magn),
            wdsz,
            %Val(thin) )

C
C      ** thin the edges **
C

type * , ' ** Call THINEDGES ** '

Call THINEDGES ( %Val(magn),      ! magnitude of edge gradient
                 %Val(phas),      ! phase of edge gradient
                 wdsz,           ! size of subimages
                 %Val(thin) )    ! thinned edges

file = cfil(1:lastpos(cfil)-4)//'.thn'
Call VIMPUT ( %Val(thin) , file , 1 , stat )

```

```

type * , ' File created ==> ' , file

C
C
C      ** threshold thinned edges **

Call VIMZER ( %Val ( thrs ) , byt2 ) ! initialize threshold array
type * , ' ** Call APSTHRES ** '

Call APSTHRES ( %Val(magn),      ! edge magnitude
.               %Val(thin),      ! thinned edges
.               wdsz,            ! size of subimages
.               mask,            ! size of mask used for gradient
.               fact,            ! threshold formula factor
.               trs2,            ! threshold used later for relaxation
.               %Val(thrs) )      ! thresholded thinned edges

file = cfil(1:lastpos(cfil)-4)//'.trs'
Call VIMPUT ( %Val (thrs) , file , 1 , stat )
type * , ' File created ==> ' , file

C
C
C      -- initialize 'rlax' with 'thrs' --

type * , ' ** Call SAVE ** '

Call SAVE ( %Val(thrs),
.           wdsz,
.           %Val(rlax) )

C
C
C      ** lower threshold for those points in thic **

type * , ' ** Call RELAX ** '

Call RELAX ( %Val(thin),      ! non-binary thinned edges
.           %Val(thic),      ! guide for location of relaxed thrs
.           lbnd,            ! guide for number of images in 'thic'
.           wdsz,            ! size of subimages
.           trs2,            ! relaxed threshold
.           %Val(rlax) )      ! thrs with threshold relaxed

file = cfil(1:lastpos(cfil)-4)//'.rlx'
Call VIMPUT ( %Val (rlax) , file , 1 , stat )
type * , ' File created ==> ' , file

C
C
C      -- initialize 'clen' with 'rlax' --

type * , ' ** Call SAVE ** '

Call SAVE ( %Val(rlax),
.           wdsz,
.           %Val(clen) )

C

```

```

c      ** clean little glitches before finding endpoints **
c
type * , ' ** Call CLEAN ** '
Call CLEAN ( %Val(rfax),      ! relaxed thresholded edges
             wdsz,           ! size of subimage
             %Val(cfen) )    ! cleaned edges

file = cfil(1:lastpos(cfil)-4)//'.cln'
Call VIMPUT ( %Val (cfen) , file , 1 , stat )
type * , ' File created ==> ' , file

c
c      -- initialize 'nend' with 'cfen' --
c

type * , ' ** Call SAVE ** '
Call SAVE ( %Val(cfen),
            wdsz,
            %Val(nend) )

c
c      ** find endpoints **
c

type * , ' ** Call ENDPTS ** '
Call ENDPTS ( %Val(cfen),      ! thinned and thresholded edges
             wdsz,           ! size of subimage
             %Val(nend) )    ! edges with endpts marked

file = cfil(1:lastpos(cfil)-4)//'.end'
Call VIMPUT ( %Val (nend) , file , 1 , stat )
type * , ' File created ==> ' , file

c
c      -- initialize 'conn' with 'nend' --
c

type * , ' ** Call SAVE ** '
Call SAVE ( %Val(nend),
            wdsz,
            %Val(conn) )

c
c      ** preserve edges if thry were in past edge image **
c

type * , ' ** Call PRESERVE ** '
Call PRESERVE ( %Val(nend),    ! edges with endpts marked
               %Val(temp),    ! final edges last frame
               wdsz,          ! size of subimage
               %Val(conn) )   ! final edges with gaps linked

file = cfil(1:lastpos(cfil)-4)//'.con'
Call VIMPUT ( %Val (conn) , file , 1 , stat )

```

```

type * , ' File created ==> ' , file

C
C   -- save 'clen' for use next round in PRESERVE --
C

type * , ' ** Call SAVE ** '

Call SAVE ( %Val(clen),
.           wdsz,
.           %Val(temp) )

C
C           ** fillin edges to get segmented object **
C

type * , ' ** Call FILLIN ** '

Call FILLIN ( %Val(conn),      ! fillin final edges
.             wdsz,            ! subimages size
.             8,               ! search directions
.             255,             ! fillin intensity
.             %Val(fill) )     ! edges filled

C
C           ** segment target from subimage **
C

Call AND ( %Val(sub2),        ! target subimage
.          %Val(fill),        ! filled edges
.          wdsz,              ! subimage size
.          %Val(segm) )       ! segmented target

file = cfil(1:lastpos(cfil)-4)//'.seg'
Call VIMPUT ( %Val (segm) , file , 1 , stat )
type * , ' File created ==> ' , file

C
C           ** sum thresholded edges **
C

type * , ' ** Call SUM ** '

Call SUM ( %Val(rlax),        ! edges after threshold relaxation
.          wdsz,              ! size of subimages
.          lbnd,              ! guide for number of images to sum
.          %Val(thic) )       ! thick edges produced by summing

file = cfil(1:lastpos(cfil)-4)//'.sum'
Call VIMPUT ( %Val (thic) , file , 1 , stat )
type * , ' File created ==> ' , file

C
C           -- load 'img2' with next image in sequence --
C

encode(16,100,cfil) 'seq', seqn, 'f', fram, '.img'

Call VIMALL ( cfil, hdr2, img2, vsiz, hsiz, 1, 'mod', stat )

```

```
If ( .not. stat ) Call EXIT ( stat )  
type * , ' Present frame is in file ==> ',cfil  
end do  
type * , ' TEBS ==> Q.E.D. '  
stop  
end
```

```

c
c-----
c
c      Subroutine EXTRACT
c
c      Purpose:      To extract a subimage using a given center and
c                    desired size.
c
c      Inputs:       img1      large image
c                    hsiz      horizontal size of large image
c                    vsiz      vertical size of large image
c                    wdsz      size of subimage
c                    crow      center row of subimage
c                    ccol      center column of subimage
c
c      Outputs:      sub1      output subimage
c-----
c
c      Subroutine EXTRACT ( img1, hsiz, vsiz, wdsz, crow, ccol, sub1 )
c
c          -- declare all variables --
c
c      Implicit      none
c
c      Integer*2      row , col
c      Integer*2      crow , ccol
c      Integer*2      rndx , cndx
c      Integer*2      hsiz , vsiz , wdsz
c      Integer*2      left , right , up , down
c      Integer*2      img1 ( hsiz , vsiz )
c      Integer*2      sub1 ( wdsz , wdsz )
c
c          -- save subimage centered around given coords. --
c
c      left = crow - wdsz/2      ! calculate sides of subimage
c      right = crow + wdsz/2
c      up = ccol - wdsz/2
c      down = ccol + wdsz/2
c
c      rndx = 0      ! initialize row index
c
c      do row = left, right - 1
c
c          cndx = 0      ! initialize col index
c          rndx = rndx + 1      ! increment row index for subimage
c
c          do col = up, down - 1
c
c              cndx = cndx + 1      ! increment col index for subimage
c
c              sub1 ( cndx, rndx ) = img1 ( col, row )
c
c          end do
c      end do
c
c      return
c      end

```

```

C
C-----
C
C      Subroutine SAVE
C
C      Purpose:      To duplicate an image array into another
C                    file for initialization or to keep for
C                    further processing.
C
C      Inputs:       inpt      image to be duplicated
C                    wdsz      size of image
C
C      Outputs:      otpt      output image ( same as input )
C-----
C
C
C      Subroutine SAVE ( inpt , wdsz , otpt )
C
C      -- declare all variables --
C
C      Implicit      none
C
C      Integer*2      wdsz
C      Integer*2      row , col
C      Integer*2      inpt ( wdsz , wdsz )
C      Integer*2      otpt ( wdsz , wdsz )
C
C      -- duplicate input array into output file --
C
C      do row = 1 , wdsz
C        do col = 1 , wdsz
C
C          otpt (col, row) = inpt (col, row)
C
C        end do
C      end do
C
C      return
C      end

```

```

c
c-----
c
c      Subroutine CORR
c
c      Purpose:      To perform simple additive difference correlation
c                    in order to find any offset needed to register two
c                    consecutive frames in a sequence.
c
c      Inputs:       sub1      past subimage of target
c                    img2      current scene image
c                    hsiz      horizontal size of large image
c                    vsiz      vertical size of large image
c                    wdsz      size of subimage
c                    crow      center row of target
c                    ccol      center column of target
c
c      Outputs:      sub2      new subimage of target
c
c-----
c
c      Subroutine CORR ( sub1, img2, hsiz, vsiz, wdsz,
c                      crow, ccol, sub2 )
c
c      -- declare all variables --
c
c      Implicit      none
c
c      Integer*2      delta
c      Integer*2      row , col
c      Integer*2      crow , ccol
c      Integer*2      rndx , cndx
c      Integer*2      xdel , ydel
c      Integer*2      xoff , yoff
c      Integer*2      hsiz , vsiz , wdsz
c      Integer*2      left , right , up , down
c      Integer*2      sub1 ( wdsz , wdsz )
c      Integer*2      img2 ( hsiz , vsiz )
c      Integer*2      sub2 ( wdsz , wdsz )
c
c      Integer*4      min
c      Integer*4      sum ( -2:2 , -2:2 )
c
c      -- parameters --
c
c      delta = 2      ! max expected object shift
c      min = 1000000  ! initialize correlation minimum
c
c      -- sum differences for correlation factor --
c
c      do xdel = -delta , delta      ! shift a predetermined amount
c        do ydel = -delta , delta    ! to catch max. movement
c
c          left = crow - wdsz/2 + xdel ! correlate over a subimage

```

```

right = crow + wdsz/2 + xdel      ! with boundaries given by
up    = ccol - wdsz/2 + ydel      ! target center and size.
down  = ccol + wdsz/2 + ydel      ! shift over delta window

sum ( xdel , ydel ) = 0           ! initialize sum
rndx = 0                          ! initialize row index

do row = left , right - 1

    rndx = rndx + 1                ! increment row index
    cndx = 0                      ! initialize column index

    do col = up , down - 1

        cndx = cndx + 1           ! increment column index

        ! correlation is simply the absolute sum of differences

        sum ( xdel , ydel ) = sum ( xdel , ydel ) +
                                abs ( subl ( cndx , rndx ) -
                                              img2 ( col , row ) )

    end do
end do

if ( sum(xdel, ydel) .lt. min ) then ! save offset for
    xoff = xdel                     ! best correlation
    yoff = ydel                     ! (i.e. min. difference)
    min = sum(xdel,ydel)
end if

end do
end do

type *, ' offsets ( x , y ) ==> ',xoff,yoff

c
c      -- update center row and column --
c

type *, ' old center = ',crow,ccol

crow = crow + xoff
ccol = ccol + yoff

type *, ' new center = ',crow,ccol

c
c      -- save subimage with target centered --
c

left  = crow - wdsz/2              ! save the subimage given by the
right = crow + wdsz/2              ! previous center corrected by
up    = ccol - wdsz/2              ! the offset just calculated
down  = ccol + wdsz/2

rndx = 0                          ! initialize row index

do row = left, right - 1

```

```

cndx = 0                                ! initialize col index
rndx = rndx + 1                          ! increment row index for subimage

do col = up, down - 1
    cndx = cndx + 1                      ! increment col index for subimage
    sub2 ( cndx, rndx ) = img2 ( col, row )
end do
end do

return
end

```

```

C
C-----
C
C      Subroutine ALPHA
C
C      Purpose:      To implement the alpha filter to reduce
C                     the noise characteristics of a sequence
C                     of images.
C
C      Inputs:        sub1      past averaged image
C                     sub2      subimage from present scene
C                     wdsz      size of target subimage
C                     alfa      filter feedback coeff.
C
C      Outputs:       avrg      new averaged image
C-----
C
C      Subroutine ALPHA ( sub1, sub2, wdsz, alfa, avrg )
C
C      Alpha filter formula :
C
C          filtered = image(t-1) * ( alpha ) + image(t) * ( 1 - alpha )
C                  = image(t) + alpha * ( image(t-1) - image(t) )
C
C          -- declare all variables --
C
C      Implicit      none
C
C      Integer*2      nint
C      Integer*2      byt2
C      Integer*2      wdsz
C      Integer*2      row , col
C      Integer*2      sub1 ( wdsz , wdsz )
C      Integer*2      sub2 ( wdsz , wdsz )
C      Integer*2      avrg ( wdsz , wdsz )
C
C      Real*4         alfa
C
C          -- alpha filter the images --
C
C      do row = 1 , wdsz
C        do col = 1 , wdsz
C
C          -- use alpha to combine two alined images --
C
C          avrg (col, row) = sub2 (col, row) + nint ( alfa * (
C              sub1(col, row)-sub2(col, row)))
C
C        end do
C      end do
C
C      return
C      end

```

```

C
C-----
C
C      Subroutine PREWIT
C
C      Purpose:      To pass a Prewitt Mask over an image in order
C                    to obtain an edge gradient magnitude and an
C                    edge phase image.
C
C      Inputs:       inpt      input image with target centered
C                    wdsz      size of target image
C                    mask      mask size of Prewitt Gradient
C
C      Outputs:      magn      magnitude of edge image
C                    phas      phase of edge image
C-----
C
C      Subroutine PREWIT ( inpt, wdsz, mask, magn, phas )
C
C
C          Prewitt Gradient
C
C          -1 0 1      -1 -1 -1
C          -1 0 1      0 0 0
C          -1 0 1      1 1 1
C
C          vertical      horizontal
C
C      Normalize the sqrt of the sum of the squares of these two gradients
C      to obtain the magnitude of the Prewitt Gradient.
C
C          -- declare all variables --
C
C      Implicit      none
C
C      Integer*2      wdsz
C      Integer*2      iang
C      Integer*2      mask , hmsk
C      Integer*2      left , right , up , down
C      Integer*2      row , row_index , col , col_index
C      Integer*2      inpt ( wdsz, wdsz )
C      Integer*2      magn ( wdsz, wdsz )
C      Integer*2      phas ( wdsz, wdsz )
C
C      Real*4          h_grad , v_grad , norm , pi , rang , conv
C
C          -- set up parameters --
C
C      Pi = 4 * Atan ( 1. )
C      Conv = 180 / Pi      ! conversion factor
C      hmsk = mask / 2      ! dist from center of mask
C      norm = mask**2 / 2    ! normalize the two square masks

```

```

C
C      -- calculate Prewitt Gradient --
C
do row = 1 + hmsk , wdsz - hmsk
  do col = 1 + hmsk , wdsz - hmsk

    v_grad = 0.0                                ! initialize vertical grad.
    do row_index = -hmsk , hmsk                ! scan rows
      do col_index = 1 , hmsk                  ! distance from center col
        left  = inpt ( col - col_index , row + row_index )
        right = inpt ( col + col_index , row + row_index )
        v_grad = v_grad + ( right - left ) ! mask sum
      end do
    end do

    h_grad = 0.0                                ! initialize horiz. grad.
    do col_index = -hmsk , hmsk                ! scan cols
      do row_index = 1 , hmsk                  ! distance from center row
        up    = inpt ( col + col_index , row - row_index )
        down  = inpt ( col + col_index , row + row_index )
        h_grad = h_grad + ( down - up ) ! mask sum
      end do
    end do

C
C      -- normalize and clip mask gradient --
C
    magn ( col , row ) =
      min ( 255.0 , sqrt ( h_grad**2 + v_grad**2 ) / norm )

C
C      -- calculate phase of gradient --
C
C Phase convention:
C
C      Cold      Hot      Cold
C
C      2 /      8 \
C      3 |      | 7
C      4 \      6 /
C      5
C
    if ( magn ( col , row ) .gt. 0. ) then
      ! calculate angle from directional gradients

```

```

        Rang = Atan2 ( v_grad , h_grad ) * Conv
! all angles made between 0 and 360 degrees
        if ( Rang .lt. 0 ) Rang = Rang + 360.
! round off angle to nearest 45 degrees
        Iang = ( Rang + 22.5 ) / 45.
! phase is a code depending in angle
        Phas ( col , row ) = mod ( Iang , 8 ) + 1
    end if
end do
end do
return
end

```



```

        if ( phas(col,row).eq.1 .or. phas(col,row).eq.5 ) then
c
c
c
            -- horizontal edge --

            if ( magn ( col, row-1 ) .gt. magn ( col, row ) .or.
              .   magn ( col, row+1 ) .gt. magn ( col, row ) )
              .   thin ( col , row ) = 0

            else if ( phas(col,row).eq.3 .or. phas(col,row).eq.7 ) then

c
c
c
                -- vertical edge --

                if ( magn ( col-1, row ) .gt. magn ( col, row ) .or.
                  .   magn ( col+1, row ) .gt. magn ( col, row ) )
                  .   thin ( col , row ) = 0

                else if ( phas(col,row).eq.2 .or. phas(col,row).eq.6 ) then

c
c
c
                    -- right diagonal edge --

                    if ( magn ( col-1, row-1 ) .gt. magn ( col, row ) .or.
                      .   magn ( col+1, row+1 ) .gt. magn ( col, row ) )
                      .   thin ( col , row ) = 0

                    else if ( phas(col,row).eq.4 .or. phas(col,row).eq.8 ) then

c
c
c
                        -- left diagonal edge --

                        if ( magn ( col-1, row+1 ) .gt. magn ( col, row ) .or.
                          .   magn ( col+1, row-1 ) .gt. magn ( col, row ) )
                          .   thin ( col , row ) = 0

                        end if
                    end do
                end do
            return
        end

```

```

C
C-----
C
C      Subroutine APSTHRES
C
C      Purpose:      To calculate a threshold based on the statistical
C                    properties of mean and standard deviation and apply
C                    this threshold to an image.
C
C      Inputs:      magn      edge magnitude image
C                   thin      thinned edge image
C                   wdsz      target image size
C                   mask      mask size used on edge gradient
C                   fact      threshold factor
C
C      Outputs:     trs2      relaxed threshold used later
C                   thrs      thresholded image
C-----
C

      Subroutine APSTHRES ( magn , thin , wdsz ,
                           mask , fact , trs2 , thrs )

C
C      -- declare all variables --
C
      Implicit      None
      Integer*2     fact
      Integer*2     taps
      Integer*2     trs2
      Integer*2     wdsz
      Integer*2     row , col
      Integer*2     mask , hmsk
      Integer*2     magn ( wdsz , wdsz )
      Integer*2     thin ( wdsz , wdsz )
      Integer*2     thrs ( wdsz , wdsz )

      Integer*4     num_pix

      Real*8        stdv
      Real*8        sum1 , sum2
      Real*8        mean1 , mean2

C
C      -- set up parameters --
C
      hmsk = mask / 2      ! dist from center of mask

      sum1 = 0
      sum2 = 0

C
C      -- find sum and sum of squares --
C
      do row = 1 + mask , wdsz - mask

```

```

        do col = 1 + mask , wdsz - mask
            sum1 = sum1 + magn ( col , row )
            sum2 = sum2 + magn ( col , row ) ** 2
        end do
    end do

C
C        -- total number of pixels --
C

num_pix = 1. * ( wdsz - 2*mask ) * ( wdsz - 2*mask )

C
C        -- use these sums to find standard deviation --
C

mean1 = sum1 / num_pix
mean2 = sum2 / num_pix
stdv = sqrt ( mean2 - mean1 ** 2 )

C
C        -- calc threshold using mean and standard deviation --
C

taps = mean1 + fact * ( stdv / ( 1. - 1 / mean1 ) )
trs2 = taps / 2                ! relaxed threshold for later use

C
C        -- print results --
C

type *, ' mean ==> ' , mean1
type *, ' stdv ==> ' , stdv
type *, ' taps ==> ' , taps
type *, ' trs2 ==> ' , trs2

C
C        -- threshold thinned edges image --
C

do row = 1 + mask , wdsz - mask
    do col = 1 + mask , wdsz - mask

        if ( thin ( col , row ) .ge. taps ) then
            thrs ( col , row ) = 255
        end if

    end do
end do

return
end

```

```

c
c-----
c
c      Subroutine RELAX
c
c      Purpose:      To relax the threshold applied to the edge image
c                    in hopes of finding more edge points in specified
c                    locations.
c
c      Inputs:       thin    thin edge image
c                   thic    thick edge image
c                   trs2    relaxed threshold
c                   lbnd    lower bound for object points in thic
c                   wdsz    size of subimage
c
c      Outputs:      thrs    thresholded edges relaxed in specified region
c-----
c
c      Subroutine RELAX ( thin , thic , lbnd , wdsz , trs2 , thrs )
c
c      -- declare all variables --
c
c      Implicit      none
c
c      Integer*2     wdsz
c      Integer*2     lbnd
c      Integer*2     trs2
c      Integer*2     row , col
c      Integer*2     row1 , col1 , row2 , col2
c      Integer*2     thin ( wdsz , wdsz )
c      Integer*2     thic ( wdsz , wdsz )
c      Integer*2     thrs ( wdsz , wdsz )
c
c      -- threshold thinned edges image with new lower threshold --
c
c      do row = 1 , wdsz
c        do col = 1 , wdsz
c
c          if ( thin ( col , row ) .ge. trs2 .and.
c              thic ( col , row ) .ge. lbnd ) then
c
c            thrs ( col , row ) = 255
c
c          end if
c
c        end do
c      end do
c
c      return
c      end

```

```

C
C-----
C
C      Subroutine CLEAN
C
C      Purpose:      To clean edges of small defects.
C
C      Inputs:      inpt      input edge image
C                   wdsz      size of subimage
C
C      Outputs:      otpt      output cleaned edge image
C-----
C

      Subroutine CLEAN ( inpt, wdsz , otpt )

C
C      -- declare all variables --
C

      Implicit      none

      Integer*2      wdsz
      Integer*2      totl , sum
      Integer*2      row , col
      Integer*2      N , S , E , W , NE , SE , NW , SW
      Integer*2      inpt ( wdsz , wdsz )
      Integer*2      otpt ( wdsz , wdsz )

C
C      -- set up mask points as binary --
C

      do row = 2 , wdsz - 1
        do col = 2 , wdsz - 1

          if ( inpt (col , row-1) .gt. 0 ) then
            N = 1
          else
            N = 0
          end if

          if ( inpt (col+1, row-1) .gt. 0 ) then
            NE = 1
          else
            NE = 0
          end if

          if ( inpt (col-1, row-1) .gt. 0 ) then
            NW = 1
          else
            NW = 0
          end if

          if ( inpt (col , row+1) .gt. 0 ) then
            S = 1
          else
            S = 0
          end if
        end do
      end do

```

```

        if ( inpt (col+1, row+1) .gt. 0 ) then
            SE = 1
        else
            SE = 0
        end if

        if ( inpt (col-1, row+1) .gt. 0 ) then
            SW = 1
        else
            SW = 0
        end if

        if ( inpt (col+1, row ) .gt. 0 ) then
            E = 1
        else
            E = 0
        end if

        if ( inpt (col-1, row ) .gt. 0 ) then
            W = 1
        else
            W = 0
        end if

c
c      -- adding points --
c

if ( inpt(col,row) .eq. 0 ) then      ! if not object point

c
c      -- fill simple gaps --
c

        totl = N + S + E + W + NE + SE + NE + NW      ! total sum

        sum = N + S
        if ( sum .eq. 2 .and. totl .le. 4 ) then!      XX
            otpt ( col, row ) = 255                    !      --
        end if                                          !      XX

        sum = E + W
        if ( sum .eq. 2 .and. totl .le. 4 ) then!      XX -- XX
            otpt ( col, row ) = 255                    !
        end if                                          !

        sum = SE + NW
        if ( sum .eq. 2 .and. totl .le. 4 ) then!      XX
            otpt ( col, row ) = 255                    !      --
        end if                                          !      XX

        sum = NE + SW
        if ( sum .eq. 2 .and. totl .le. 4 ) then!      XX
            otpt ( col, row ) = 255                    !      --
        end if                                          !      XX

```

```

sum = N + SW
if ( sum .eq. 2 .and. totl .le. 4 ) then! XX
    otpt ( col, row ) = 255 --
end if XX

sum = N + SE
if ( sum .eq. 2 .and. totl .le. 4 ) then! XX
    otpt ( col, row ) = 255 --
end if XX

sum = S + NW
if ( sum .eq. 2 .and. totl .le. 4 ) then! XX
    otpt ( col, row ) = 255 --
end if XX

sum = S + NE
if ( sum .eq. 2 .and. totl .le. 4 ) then! XX
    otpt ( col, row ) = 255 --
end if XX

sum = E + SW
if ( sum .eq. 2 .and. totl .le. 4 ) then! XX
    otpt ( col, row ) = 255 -- XX
end if XX

sum = E + NW
if ( sum .eq. 2 .and. totl .le. 4 ) then! XX
    otpt ( col, row ) = 255 -- XX
end if XX

sum = W + NE
if ( sum .eq. 2 .and. totl .le. 4 ) then! XX
    otpt ( col, row ) = 255 -- XX
end if XX

sum = W + SE
if ( sum .eq. 2 .and. totl .le. 4 ) then! XX
    otpt ( col, row ) = 255 -- XX
end if XX

```

c
c
c

-- deleting points --

else

c
c
c

-- check for lone point --

totl = N + S + E + W + NE + SE + NE + NW ! total sum

```

        if ( totl .eq. 0 ) then
            otpt ( col, row ) = 0
        end if
    !  -- -- --
    !  -- XX --
    !  -- -- --

c
c
c
    -- check for pimple point --

    sum = NW + N + NE

    if ( sum .eq. 3 .and. totl .eq. 3 ) then
        otpt ( col, row ) = 0
    end if
    !  XX XX XX
    !  -- XX --
    !  -- -- --

    sum = NE + E + SE

    if ( sum .eq. 3 .and. totl .eq. 3 ) then
        otpt ( col, row ) = 0
    end if
    !  -- -- XX
    !  -- XX XX
    !  -- -- XX

    sum = SW + S + SE

    if ( sum .eq. 3 .and. totl .eq. 3 ) then
        otpt ( col, row ) = 0
    end if
    !  -- -- --
    !  -- XX --
    !  XX XX XX

    sum = NW + W + SW

    if ( sum .eq. 3 .and. totl .eq. 3 ) then
        otpt ( col, row ) = 0
    end if
    !  XX -- --
    !  XX XX --
    !  XX -- --

end if

    end do
end do

return
end

```

```

C
C-----
C
C      Subroutine ENDPTS
C
C      Purpose:      To find the endpoints of a thinned edge image.
C
C      Inputs:        thrs      thinned thresholded edges
C                     wdsz      target image size
C
C      Outputs:       nend      processed edges with endpoints marked
C-----
C
C      Subroutine ENDPTS ( thrs, wdsz, nend )
C
C          -- declare all variables --
C
C      Implicit      none
C
C      Integer*2      wdsz
C      Integer*2      row , col
C      Integer*2      left , right , up , down
C      Integer*2      thrs ( wdsz, wdsz )
C      Integer*2      nend ( wdsz, wdsz )
C
C          -- use mask to determine if point is an endpoint --
C
C
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C      |--|--|
C
C      endpoint if XX is object and none other in mask is object
C
C      do row = 2 , wdsz - 1
C        do col = 2 , wdsz - 1
C
C          left = 0
C          right = 0
C          up = 0
C          down = 0
C
C          ! initialize mask sums
C
C          if (thrs(col,row).eq.255) then
C            up = thrs (col-1, row-1)

```

```

.      + thrs (col , row-1)
.      + thrs (col+1, row-1)
.      + thrs (col-1, row )
.      + thrs (col+1, row )

      if ( up .eq. 0 ) then
            nend ( col, row ) = 127      ! mark as endpt
            goto 100
      end if

      down=  thrs (col-1, row+1)
.      + thrs (col , row+1)
.      + thrs (col+1, row+1)
.      + thrs (col-1, row )
.      + thrs (col+1, row )

      if ( down .eq. 0 ) then
            nend ( col, row ) = 127
            goto 100
      end if

      right=  thrs (col+1, row-1)
.      + thrs (col+1, row )
.      + thrs (col+1, row+1)
.      + thrs (col , row-1)
.      + thrs (col , row+1)

      if ( right .eq. 0 ) then
            nend ( col, row ) = 127
            goto 100
      end if

      left=  thrs (col-1, row-1)
.      + thrs (col-1, row )
.      + thrs (col-1, row+1)
.      + thrs (col , row-1)
.      + thrs (col , row+1)

      if ( left .eq. 0 ) then
            nend ( col, row ) = 127
            goto 100
      end if

100      end if

      end do
      end do

      return
      end

```

```

c
c-----
c      Subroutine PRESERVE
c
c      Purpose :      To preserve edge information in sequential
c                      frames using edges from the previous frame.
c
c      Inputs :      psnt      edge image with endpts marked
c                      past      past frame edges
c                      wdsz      size of subimages
c
c      Outputs :      conn      connected edges
c-----
c
c      Subroutine PRESERVE ( psnt , past , wdsz , conn )
c
c      -- declare all variables --
c
c      Implicit      none
c
c      Integer*2      delta
c      Integer*2      wdsz , csiz
c      Integer*2      row , col
c      Integer*2      irow , icol
c      Integer*2      rndx , cndx
c      Integer*2      roff , coff
c      Integer*2      i , j , k , l
c      Integer*2      psnt ( wdsz , wdsz )
c      Integer*2      past ( wdsz , wdsz )
c      Integer*2      conn ( wdsz , wdsz )
c
c      Integer*4      max
c      Integer*4      sum
c
c      -- parameters --
c
c      csiz = 8      ! correlation window size
c      delta = 1      ! look delta from center for correlation match
c
c      -- look through image for endpts --
c
c      do row = 2 + csiz/2 , wdsz - csiz/2 - 2
c        do col = 2 + csiz/2 , wdsz - csiz/2 - 2
c
c          if ( psnt ( col , row ) .eq. 127 ) then      ! check if endpt
c            type * , ' endpt found at (row,col) ==> ',row,col
c
c            max = 0      ! initialize max
c            roff = 0      ! initialize row offset
c            coff = 0      ! initialize column offset

```

```

do i = -delta, delta      ! correlate over window delta x delta
  do j = -delta, delta
    sum = 0                ! initialize sum

    do k = -csiz/2, csiz/2 ! correlation window is csiz x csiz
      do l = -csiz/2, csiz/2
! present image row index varies only with correlation window

        irow = row + k
        icol = col + l

! past image row index depends on present value of delta and window index

        rndx = row + k + 1
        cndx = col + l + j

        if ( psnt ( icol, irow ) .ne. 0 ) then      ! if object point
! if overlap ==> add 3
          if ( past ( cndx, rndx ) .ne. 0 ) sum = sum + 3
! if adjacent ==> add 1
          if(past ( icol-1, irow ) .ne. 0 ) sum = sum + 1
          if(past ( icol+1, irow ) .ne. 0 ) sum = sum + 1
          if(past ( icol-1, irow-1 ) .ne. 0 ) sum = sum + 1
          if(past ( icol , irow-1 ) .ne. 0 ) sum = sum + 1
          if(past ( icol+1, irow-1 ) .ne. 0 ) sum = sum + 1
          if(past ( icol-1, irow+1 ) .ne. 0 ) sum = sum + 1
          if(past ( icol , irow+1 ) .ne. 0 ) sum = sum + 1
          if(past ( icol+1, irow+1 ) .ne. 0 ) sum = sum + 1
        end if
      end do
    end do

    type * , ' sum (',i,j,') = ',sum

! find maximum sum ( best correlation ) and remember offset
    if ( sum .gt. max ) then
      type * , ' update offsets '

      roff = j
      coff = i
      max = sum

      type * , ' roff,coff,max ==> ',roff,coff,max

```

```

end if

      end do
end do

type * , ' row offset = ', roff
type * , ' column offset = ', coff

! add edges from past frame to present frame

type * , ' sum subwindows in frames'

do k = row - csiz/2, row + csiz/2
  do l = col - csiz/2, col + csiz/2

    conn ( l, k ) = psnt ( l, k ) +
                    past ( l + coff, k + roff )

    if ( conn ( l, k ) .gt. 255 ) conn ( l, k ) = 255

  end do
end do

end if

      end do
end do

return
end

```

```

C
C-----
C
C      Subroutine FILLIN
C
C      Purpose:      To fill in boundaries of target.
C                    (Code from M. Burton)
C
C      Inputs:      edge      input edge image
C                   siz      size of image
C                   bnds      number of neighbors ( 4 or 8 )
C                   colr      Intensity of fillin
C
C      Outputs:      fill      output image with boundaries filled
C-----
C
C      Subroutine FILLIN ( edge, size, bnds , colr , fill )
C
C      -- declare all variables --
C
C      Implicit      none
C
C      Integer*2      size,bnds,colr
C      Integer*2      edge(size,size)
C      Integer*2      fill(size,size)
C
C      Integer*2      i,j,k
C      Integer*2      ihit,imiss
C      Integer*2      i_ns,i_ew
C      Integer*2      k_n,k_s,k_e,k_w,k_nw,k_ne,k_se,k_sw
C
C      -- initialize search region --
C
C      i_ns = size
C      i_ew = size
C
C      -- initialize fill array --
C
C      do i = 1 , size
C         do j = 1 , size
C
C             fill ( j, i ) = 0
C
C         end do
C      end do
C
C      do i = 1 , size
C         do j = 1 , size
C
C             fill ( j, i ) = edge ( j, i )      ! Initialization
C
C             if ( edge(j,i) .eq. 0 ) then        ! Only check non-edge points
C
C                 ihit = 0                        !Initialize counters
C                 imiss = 0
C
C                 k_n = min(i_ns,i-1)             !Fill in North
C                 do k = 1, k_n
C                     if ( edge(j,i-k) .ne. 0 ) then

```

```

        ihit = ihit + 1
        go to 10
    end if
end do
imiss = imiss + 1
if ( imiss .gt. 8-bnds ) go to 100
10    k_s = min (i_ns,size-i)          !Fill in South
    do k = 1, k_s
        if ( edge(j,i+k) .ne. 0 ) then
            ihit = ihit + 1
            go to 20
        end if
    end do
    imiss = imiss + 1
    if ( imiss .gt. 8-bnds ) go to 100
20    k_w = min (i_ew,j-1)            !Fill in West
    do k = 1, k_w
        if ( edge(j-k,i) .ne. 0 ) then
            ihit = ihit + 1
            go to 30
        end if
    end do
    imiss = imiss + 1
    if ( imiss .gt. 8-bnds ) go to 100
30    k_e = min (i_ew,size-j)          !Fill in East
    do k = 1, k_e
        if ( edge(j+k,i) .ne. 0 ) then
            ihit = ihit + 1
            go to 40
        end if
    end do
    imiss = imiss + 1
    if ( imiss .gt. 8-bnds ) go to 100
40    k_nw = min (k_n,k_w)             !Fill in North-West
    do k = 1, k_nw
        if ( edge(j-k,i-k) .ne. 0 ) then
            ihit = ihit + 1
            go to 50
        end if
    end do
    imiss = imiss + 1
    if ( imiss .gt. 8-bnds ) go to 100
50    k_ne = min (k_n,k_e)             !Fill in North-East
    do k = 1, k_ne
        if ( edge(j+k,i-k) .ne. 0 ) then
            ihit = ihit + 1
            go to 60
        end if
    end do
    imiss = imiss + 1
    if ( imiss .gt. 8-bnds ) go to 100
60    k_se = min(k_s,k_e)              !Fill in South-East
    do k = 1, k_se
        if ( edge(j+k,i+k) .ne. 0 ) then

```

```

        ihit = ihit + 1
        go to 70
    end if
end do
imiss = imiss + 1
if ( imiss .gt. 8-bnds ) go to 100
70    k_sw = min (k_s,k_w)                !Fill in South_West
    do k = 1, k_sw
        if ( edge(j-k,i+k) .ne. 0 ) then
            ihit = ihit + 1
            go to 80
        end if
    end do
    imiss = imiss + 1
    if ( imiss .gt. 8-bnds ) go to 100
80    if ( ihit .ge. bnds ) fill(j,i) = colr/2
        end if
100    end do
end do
return
end

```

```

C
C-----
C
C      Subroutine AND
C
C      Purpose:      To extract a region from an image guided by
C                    a binary image.
C
C      Inputs:        img1      gray level intensity image
C                    img2      binary image
C                    wdsz      size of image
C
C      Outputs:       otpt      output image
C-----
C
C      Subroutine AND ( img1 , img2 , wdsz , otpt )
C
C      -- declare all variables --
C
C      Implicit      none
C
C      Integer*2      wdsz
C      Integer*2      row , col
C      Integer*2      img1 ( wdsz , wdsz )
C      Integer*2      img2 ( wdsz , wdsz )
C      Integer*2      otpt ( wdsz , wdsz )
C
C      -- output only where image2 is on --
C
C      do row = 1 , wdsz
C        do col = 1 , wdsz
C
C          if ( img2(col, row) .ne. 0 ) then
C            otpt (col, row) = img1 (col, row)
C          else
C            otpt (col, row) = 0
C          end if
C        end do
C      end do
C
C      return
C      end

```

```

C
C-----
C
C      Subroutine SUM
C
C      Purpose:      To continuously update a sum of the past N images
C                    in a sequence.
C
C      Inputs:      inpt      input image (all values = 255)
C                   wdsz      size of subimages
C                   lbnd      lower bound for object points
C
C      Outputs:     otpt      output image (values = 255 down to 'lbnd')
C-----
C
C      Subroutine SUM ( inpt , wdsz , lbnd , otpt )
C
C          -- declare all variables --
C
C      Implicit      none
C
C      Integer*2      wdsz
C      Integer*2      lbnd
C      Integer*2      row , col
C      Integer*2      inpt ( wdsz , wdsz )
C      Integer*2      otpt ( wdsz , wdsz )
C
C          -- decrement current otpt image (shift back in time) --
C
C      do row = 1 , wdsz
C        do col = 1 , wdsz
C
C          if ( otpt (col, row) .le. lbnd ) then
C            otpt (col, row) = 0
C          end if
C
C          if ( otpt (col, row) .gt. lbnd ) then
C            otpt (col, row) = otpt (col, row) - 1
C          end if
C
C        end do
C      end do
C
C          -- sum in new image (edges with value 255) --
C
C      do row = 1 , wdsz
C        do col = 1 , wdsz
C
C          otpt (col, row) = otpt (col, row) + inpt (col, row)
C
C          if ( otpt (col, row) .gt. 255 ) otpt (col, row) = 255
C
C        end do
C      end do
C
C      return
C      end

```

VITA

Charles Henry Hayden Jr. was born in Washington D.C. on November 30, 1962. He lived in Edgewater, Maryland for 13 years before moving to beautiful East Tennessee. He attended Maryville High School and graduated in 1980. He began classwork the following summer at the University of Tennessee in Knoxville, and in June, 1984 graduated with with a Bachelor of Science in Electrical Engineering with High Honors.

During his undergraduate studies, he was inducted into the Tau Beta Pi and Eta Kappa Nu Honor Societies. He was also involved in the Institute of Electrical Engineers as well as community services such as The Blount County Rescue Squad and Townsend Police Department. His undergraduate career was culminated by receiving the Texas Instruments Image Processing Fellowship in 1984.

The fellowship provided the financial resources and research facilities needed to continue education and pursue a Master of Science in Electrical Engineering. After three quarters of coursework and a summer of research at Texas Instruments in Dallas, Texas, his thesis was finalized and the degree was conferred in December 1985.