

AD-A127 409

DISPARITY ANALYSIS OF TIME-VARYING IMAGERY(U) AIR FORCE
INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF
ENGINEERING F D COOPER DEC 81 AFIT/Geo/EE/81D-1

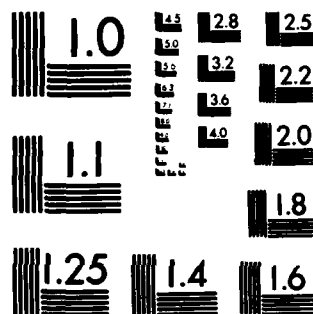
1/4

UNCLASSIFIED

F/G 20/6

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD A127409

AFIT/GEO/EE/81D-1

DISPARITY ANALYSIS OF TIME-VARYING IMAGERY

THESIS

AFIT/GEO/EE/81D-1 Franklin D. Cooper III
Capt USAF

Approved for public release; distribution unlimited.

DTIC
ELECTE
APR 28 1983

E

DISPARITY ANALYSIS OF TIME-VARYING IMAGERY

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the
Requirements for the Degree of
Masters of Science

by

Franklin D. Cooper III, B.S.
Capt USAF

Graduate Electrical Engineering

December 1981

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	

Approved for public release; distribution unlimited.



PREFACE

I selected the topic of disparity analysis for my thesis research because of the importance of this research area to the Air Force. Also, I have always been interested in the area of image processing and computer systems, and I wanted to accomplish an effort in which I could integrate these two areas into one comprehensive research project. By developing an interactive program to perform disparity analysis of time-varying imagery, I was able to expand my knowledge of both image processing systems and computer systems; and, at the same time, provide the Air Force with a valuable research tool.

I would first like to thank God for his help in all the phases of this effort. I am also indebted to Dr. Matthew S. Kabrisky, my advisor, for his guidance and support throughout this study; and to Dr. Louis A. Tamburino for his timely suggestions concerning the implementation of the interest operators and the matching algorithm, and for his assistance in obtaining the data bases used for processing. Next, I would like to thank Mr. Bruce Berley, manager of the PDP DEC 11/45 computer operations center, for the many valuable hours of assistance he rendered when troublesome computer-related systems integration problems arose. I would like to thank Mr. Charles E. Wagner for the development of the RAMTEK display algorithms used in the image display portion of the program that was developed. I would like to thank Lt. Donald L. Sander for his assistance in obtaining the

photographs of the RAMTEK image displays.

I would also like to thank Dr. Hans Nagel, Dr. William B. Thompson, Dr. Wesley Snyder, and Marsha Jo Hannah for their contributions, support, and assistance in obtaining and implementing the three data bases used in this effort. Last, but by no means least, I would like to thank my wife, Linda, for her kind, patient, and understanding support throughout this very difficult and trying experience.

Franklin D. Cooper III

Contents

	Page
Preface	ii
List of Figures	vi
List of Tables	viii
Abstract	ix
I. Introduction	1
Background	1
Problem	6
Scope	7
Assumptions	9
Approach	11
Sequence of Presentation	12
II. Program DIDA: General Description	14
Image Operations Section	18
Interesting Point Selection Section	30
Interesting Point Matching Section	56
Image Display Section	66
Image Data Bases	71
III. Interesting Point Selection Algorithms	80
Simple Variance Interest Operator	82
Directed Variance Interest Operator	103
Edged Variance Interest Operator	118
Interest Operator Evaluation	124
IV. Interesting Point Matching Algorithms	126
Label and Weight Determination	137
Initial Probabilities	148
Probability Updates	149
V. Results and Conclusions	155
Image Operations Results	155
Interest Operator Results	157
Matching Algorithm Results	158
Image Display Results	160
Conclusions	162

Contents

	Page
VI. Recommendations for Further Study	178
Image Operations Recommendations	178
Interest Operator Recommendations	178
Matching Algorithm Recommendations	180
Image Display Recommendations	181
General Recommendations	182
Bibliography	184
Appendix A: Glossary of Symbols, Variables, and Arrays	186
Appendix B: DIDA Control Program	192
Appendix C: Image Operations Section	197
Appendix D: Interesting Point Selection Section	229
Appendix E: Interesting Point Matching Section	296
Appendix F: Image Display Section	351
Appendix G: Byte-to-Word Number Conversion	365
Appendix H: NVL Data Base Tape-to-Disc Conversion Program	367
VITA	369

List of Figures

<u>Figure</u>		<u>Page</u>
1	DIDA Image Processing System Structure	3
2	DIPS Program Block Diagram	14
3	DIDA Program Block Diagram	15
4	DIDA Program Flow Diagram	18
5	Image Digitization Process	20
6	Image Row and Column Size Input Flow Diagram	23
7	DIDA Program Images Description	24
8	Logical Unit Number and File Name Input Flow Diagram	25
9	DIDA Image Operations Section Flow Diagram	31
10	DIDA Program Window Description	35
11	DIDA Program Overlay and Virtual Array Structure	39
12	Window Movement During Processing	40
13	Method of Image Input During Processing	42
14	Interesting Point Set Storage Flow Diagram	55
15	DIDA Interesting Point Selection Section Flow Diagram	57
16	Interesting Point Matching Illustration	60
17	Interesting Point Set Input Flow Diagram	63
18	DIDA Interesting Point Matching Section Flow Diagram	67
19	DIDA Image Display Section Flow Diagram	72
20	Window Movement Flow Diagram	89
21	Row Change Flow Diagram	93
22	Interesting Point Interim Storage Flow Diagram	97
23	Interesting Point Correlation Window Formation	102
24	Simple Variance Interest Operator Flow Diagram	104
25	Directed Variance Interest Operator Flow Diagram	119

26	Orientation of an 8x9 and 14x4 Set of Correlation Windows by Subroutine CWDEQ8	144
27	Relaxation-Labeling Matching Algorithm Flow Diagram	154
28	Truck Image (128x128) without Interesting Points Overwritten (TRUCK.IMG;2)	164
29	Truck Image (128x128) with 284 Interesting Points Overwritten (TRUCK.IMG;2) using the Moravec Interest Operator (IOP#2)	165
30	Truck Image (TRUCK.IMG;2) Blown Up to 512x512 without Interesting Points Overwritten	166
31	Truck Image (TRUCK.IMG;2) Blown Up to 512x512 with 284 Interesting Points Overwritten using IOP#2 (Moravec Operator)	167
32	Truck Image (TRUCK.IMG;2) Blown Up to 512x512 with 465 Interesting Points Overwritten using the Simple Variance Interest Operator (IOP#1)	168
33	Truck Image (TRUCK.IMG;2) Blown Up to 512x512 with 27 Interesting Points Overwritten using the Edged Variance Interest Operator (IOP#3)	169
34	Person Sitting in Chair Image (128x128) without Interesting Points Overwritten (PERSON.IMG;1)	170
35	Person Sitting in Chair Image (128x128) with 287 Interesting Points Overwritten (PERSON.IMG;1) using the Moravec Interest Operator (IOP#2)	171
36	Person Sitting in Chair Image (PERSON.IMG;1) Blown Up to 512x512 without Interesting Points Overwritten	172
37	Person Sitting in Chair Image (PERSON.IMG;1) Blown Up to 512x512 with 287 Interesting Points Overwritten using IOP#2 (Moravec Operator)	173
38	Person Sitting in Chair Image (PERSON.IMG;1) Blown Up to 512x512 with 59 Interesting Points Overwritten using the Edged Variance Interest Operator (IOP#3)	174
39	Image of a Taxi Cab Turning a Corner Near the University of Hamburg (512x512) (TRAFIC.IMG;1)	175
40	Image of a Section of the Night Vision Lab Terrain Board (512x512) (TERRAIN.IMG;1)	176

List of Tables

<u>Table</u>		<u>Page</u>
I	Functions, Ranges, and Specifications of Disparity Analysis Variables	10
II	Interesting Point Set File Structure	53
III	Subroutine IPSOPN Information Display	62
IV	Image Data Base Characteristics	79
V	Relative Locations of Labels and Weights in Array STORE . .	140
VI	Effect of Changing the Variance Threshold on the Number of Interesting Points Produced	159
VII	Matching Efficiencies Obtained from Various Interesting Point Sets	161
VIII	Summary of Imagery Photographs	163

ABSTRACT

An interactive program (called program DIDA) was developed to perform disparity analysis of time-varying imagery. The program consists of four sections: (1) image operations; (2) interesting point selection; (3) interesting point matching; and (4) image display. Three data bases were obtained: (1) a 20-frame sequence of images from a terrain board of the U.S. Army Night Vision Laboratory; (2) an 18-frame sequence of various moving objects from the University of Minnesota; and (3) a 33-frame sequence of images of a traffic scene from the University of Hamburg.

Each of the four program sections were tested. Image operations included printing image intensities, and image file creation and deletion. Three interest operators were employed in the interesting point selection section: (1) simple variance; (2) directed variance (the Moravec Operator); and (3) edged variance. The number of interesting points selected was found to be inversely proportional to the interest operator threshold. A relaxation-labeling matching algorithm was used in the interesting point matching section to match interesting points from two images. Very poor matching results were obtained. Image displays included black-and-white and color images with and without the interesting points overwritten. Photographs were taken of both cases.

DISPARITY ANALYSIS OF TIME-VARYING IMAGERY

I. Introduction

"For our weapons of warfare are not carnal, but mighty through God to the pulling down of strongholds."

- 2 Corinthians 10:4

An enemy tank is moving about within a heavily cluttered battle area. If the tank is being sought by an airborne munitions platform, how can it be detected, identified, and destroyed? One method is to use disparity analysis to perform the following tasks: (1) separate the moving (dynamic) objects within the battle area (scene) from the stationary (static) background; (2) isolate (segment) the moving objects from one another; (3) identify one of the moving objects as the enemy tank; and (4) track the tank as it moves from scene to scene. Some type of munitions may then be discharged along the path of movement of the tank to effect its destruction.

Background

For the past several months, the Information Processing Technology Branch of the Air Force Avionics Laboratory (AFWAL) has been actively engaged in the development of a dynamic image disparity analysis (DIDA) processing system. Under the current direction of Dr. Louis

A. Tamburino, AFWAL/AAAT-1, this development effort has moved toward the generation of a computer program for performing real-time disparity analysis of complex, time-varying imagery. This program development effort is an offspring and expansion of a program originally created at Rome Air Development Center (RADC) called IPS (for Image Processing System).

The IPS program [Ref. 1:2], in its original form, takes an image stored on an image file, performs some processing operation (or operations) on it, then places the modified image back onto the image file. The program also incorporates a data base management system for manipulation of the stored image data. Because of these capabilities, Dr. Tamburino chose the IPS program to form a basis for the DIDA processing system.

When the IPS program was obtained from RADC, it was written in MACRO Assembly Language for the DEC PDP 11/45 minicomputer. Because of the difficulties involved in using assembly language, the program was rewritten in a high level language (FORTRAN IV-PLUS) by Mr. Dug Kyoo Choi from Korea, working in this country under a scientific exchange program [Ref. 3]. He structured the program into three major divisions: (1) Data Manipulations; (2) Image Processing System; and (3) Dynamic Image Disparity Analysis. Each division was then broken up into its corresponding sub-divisions (which he called Frames) as shown in Figure 1. The overall program was entitled DIPS (for DIDA Image Processing System). During the time he was in this country, Mr. Choi

succeeded in programming and implementing the first two major sections of the DIPS program.

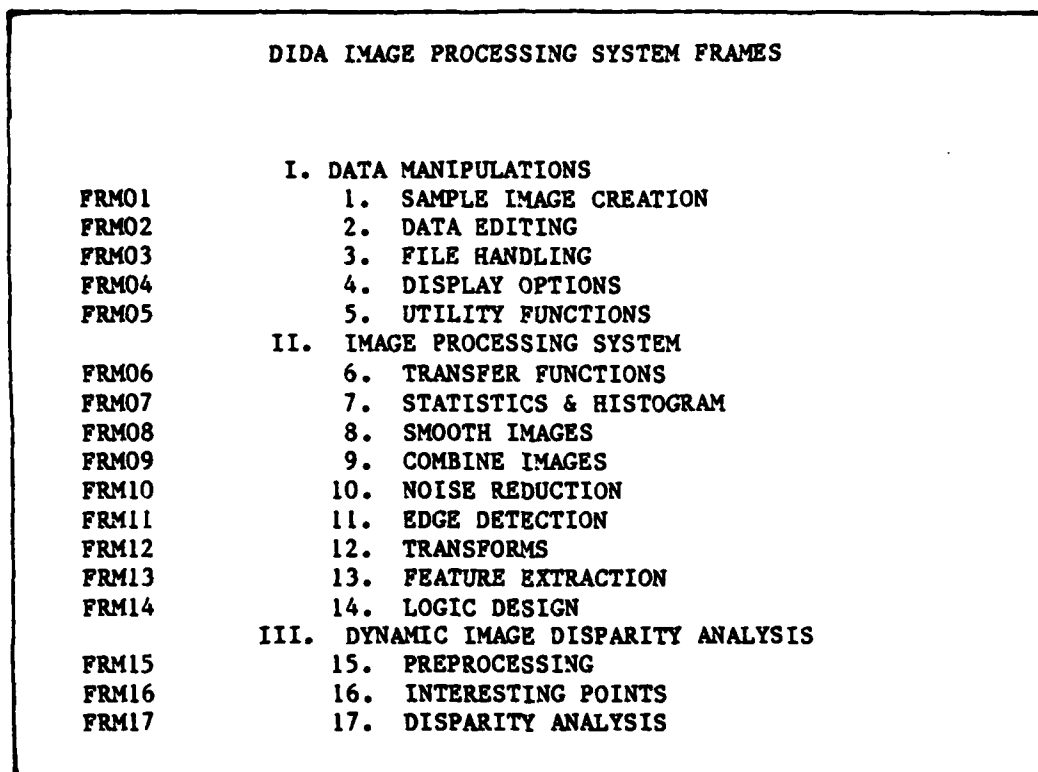


Figure 1. DIDA Image Processing System Structure [Ref.3:10]

The third section of the DIPS program forms the basis for this thesis effort. It is the portion of the DIPS program which actually performs the disparity analysis operations. This portion of the program has been entitled DIDA (for Dynamic Image Disparity Analysis).

Disparity analysis is defined to be the determination of the geometric differences between two or more images of the same or similar areas (scenes). When differences occur between two or more images, the "disparity" between them may be represented as a vector field mapping

all of the various images into the image of interest. Thus, the disparity analysis of two or more images results in the assignment of disparities, which are represented as multidimensional vector fields, to a collection of points in the image plane. These disparities are then used to make inferences about dynamic features of images [Ref. 4:333].

More simply stated, disparity analysis involves the detection and location of dynamic features of an image by the disparities created as a result of their motion. To obtain these disparities, a set of "interesting" points are formed (selected) on an image using an interest operator. Once these points are determined, they are matched to the same point in subsequent images using a matching algorithm. The difference between the location of a given interesting point in one image, and its corresponding location in subsequent images forms a disparity vector between the points. The collection of all such disparity vectors forms the vector field which maps all of the images into one.

Segmentation is the process of partitioning an image into meaningful parts such that all points belonging to a particular part have some common property (as the tank mentioned on page 1), or can be represented using a mathematical or logical predicate [Ref. 5:13]. If an object is moving within a certain scene, segmentation may be accomplished by the use of velocity information relating the time variation of the image intensity of the interesting points due to their motion to the spatial variation of the intensity over the objects

surface. A clustering algorithm is then applied to determine the dominant velocities in the range of the image space, and to classify each set of interesting points based on their estimated speed and direction [Ref 6:544].

The matching of interesting points from image to image is the very heart of disparity analysis since it forms the basis for the extraction of all pertinent information. One very important result of this matching process is the ability to delineate object boundaries as loci of interesting points. The basic idea is that a subset of the interesting points forms a skeletal template of the object within the image area. To identify the object specifically (e.g., as the tank mentioned on page 1), this skeletal template is compared with the template of the object being sought. If a reliable match occurs, then the object is searched for in a second (or subsequent) image. When located, this newly discovered template becomes the template sought in the next image (or image sequence). This process continues as long as object tracking is desired.

Because of the ability to separate static and dynamic image features, segment dynamic features from one another, and identify and track segmented dynamic features, disparity analysis forms an extremely useful tool for the detection and classification of moving objects within cluttered scenes. In the past, this type of scene analysis has posed serious problems for conventional pattern recognition algorithms. For this reason, the development of a robust, real-time dynamic image

disparity analysis processing system is of great interest to Dr. Tamburino, and the United States Air Force.

Problem

The primary objective of this thesis effort was to develop a method for performing disparity analysis of time-varying images. Given a feature of interest in a certain image, locate this feature in a subsequent image by matching the interesting points of the feature to the same interesting points in the image of interest. This involves applying an interest operator consecutively to two or more subsequent images and matching the resulting interesting points from each image.

A second goal of this thesis effort was to investigate the effects of interest operator performance on matching efficiency. For a particular interest operator, determine the matchability of the interesting points selected compared with the matching ability of other interest operators for the same image sequence. In other words, compare the ability of each interest operator to select interesting points in a fixed image sequence that are easily matched.

The final objective of this thesis effort was to develop a display algorithm to exhibit the interesting points selected by the interest operators. First, the image from which the interesting points were extracted is displayed. Next, the (x-y) coordinate location of each interesting point on the image is overwritten by some symbol which is easily distinguished from surrounding points. Finally, the image is redisplayed with the interesting points clearly visible.

Scope

From the moving tank senario on page 1, it is evident that the disparity analysis process involves four fundamental tasks: (1) to separate static and dynamic image features; (2) to segment image features moving in relation to one another; (3) to specifically identify image features (e.g., the tank); and (4) to track a specific image feature (or features) from scene to scene. This thesis effort was involved with only the first of these tasks; that is, to separate dynamic and static image features from one another by means of the disparities created from a certain image sequence. The segmentation, identification, and tracking of image features are each complex studies in their own right, and could not be adequately treated in a combined effort over the period of time allowed for this research.

Only two-image sequences were used to create disparities during this study. These sequences were taken from three separate image data bases which were provided from external sources during the effort (these data bases and sources are described more fully in Chapter II). Three different interest operators were used to select interesting points, and one matching algorithm (relaxation-labeling) was used to accomplish interesting point matching. After creation, the interesting points, along with other pertinent data, are stored on a separate external file so that the matching algorithm may be applied

independently of interesting point selection. This places no restriction on the matching algorithm as to the sets of points chosen to be matched (i.e., the images whose interesting points are being matched do not have to be exactly time sequential). Thus, any two sets of interesting points from any two images may be matched regardless of their time and feature correspondence. This is particularly useful if the testing of the resolution of the matching algorithm being used is desired.

For the disparity analysis process, image sizes in the range from 4×4 up to $512 \times (\sim 512)$ are permitted, with intensity values represented by 8-bit bytes in the range from 0 (totally dark) to 255 (totally white). For interesting point selection, the processing window size is a function of the user specified image size, with a range from 2×2 up to 63×511 permitted. Similarly, the maximum allowable number of interesting points which can be selected for a given image is a function of the maximum permissible image size, and is equal to the maximum permissible image column size specified by the system (M).

For matching, the total size (product of rows and columns) of the correlation window surrounding each interesting point for determining initial probabilities may not exceed one third of the maximum permissible image column size specified by the system, the minimum allowable size being 2×2 . The maximum sizes for the disparity window for label computation and the maximum likelihood neighborhood for probability updates are likewise a function of the maximum permissible

image column size. The permissible range of the disparity window is 2x2 up to 512x512, and the same is true for the maximum likelihood neighborhood.

Some other variables associated with both the interest operator and matching algorithms may be specified interactively during the disparity analysis process, and may take on any value within the allowable integer or floating point range of the computational system being used. In this case, it is the responsibility of the individual performing the analysis to insure that the appropriate value is specified.

Finally, for image input, the image input row buffer size is a function of the maximum permissible image size, with a range from 1x4 up to 4x512 permitted. All of these variables are discussed more fully in Chapter II. The functions, ranges, and specifications for the above-mentioned variables are summarized in Table I.

Assumptions

The following assumptions were made based on an analysis of the present state-of-the-art of disparity analysis:

1. The interest operators presented in the literature will perform in the manner described. In other words, the algorithms will select interesting points as described when they are implemented on a digital computer.

Table I. Functions, Ranges, and Specifications of Disparity Analysis Variables

Disparity Analysis Variable	Function	Minimum Size	Maximum Size	Allowed User Specification ²
1. IMAGE	Specify Intensity	4 X 4	M X (N M)	$4 \leq X_i \leq M$ $4 \leq Y_i \leq N$
2. INTERESTING POINT SELECTION a. Processing Window	Select Interesting Points	2 X 2	(2N-1) X (M-1)	$2 \leq X_w \leq X_i - 1$ $2 \leq Y_w \leq Y_i - 1$
3. INTERESTING POINT MATCHING a. Correlation Window	Determine Initial Probabilities	2 X 2	M/3 (X _C Y _C)	$2 \leq X_c \leq M/6$ $2 \leq Y_c \leq M/3X_c$
b. Disparity Window	Compute Labels	2 X 2	M X M	$2 \leq X_D \leq M$ $2 \leq Y_D \leq M$
c. Maximum Likelihood Neighborhood	Update Probabilities	2 X 2	M X M	$2 \leq X_N \leq M$ $2 \leq Y_N \leq M$
4. IMAGE ROW INPUT BUFFER	Image Row Storage Buffer	1 X 4	L X M	$X_B = X_i$ $Y_B = L$
5. PROCESSING PARAMETERS a. L	Image Input Buffer Row Size	1	³ 4	—
b. M	Maximum Image Column Size	4	³ 512	—
c. N	Maximum Initial Image Input Row Size	4	³ 64	—
NOTES:				
1. Specified at compile time		3. Computational system memory size dependent		
2. Specified during program execution		4. System specified during program execution		
		5. Value may exceed M		

2. The matching algorithms presented in the literature will perform in the manner described.

3. The image data bases received from external sources contain imagery of sufficient quality and resolution (except where stated otherwise) to yield meaningful results when processed.

4. The first two sections of the DIDA Image Processing System computer program operate as stated.

5. Computer facilities and equipment necessary to implement, debug, and test the disparity analysis program were available on a full time basis (excluding normal maintenance down-time).

6. An operational program for matching sets of interesting points would be provided for use.

7. Assistance from staff personnel at AFWAL/AAAT-1 would be available in the development of the image display algorithms.

8. Evaluation criteria exist for assessing the performance of interest operators with respect to matching efficiency.

Approach

The effort began with a literature review to determine the background and history of past disparity analysis efforts, as well as the present state-of-the-art. After this, a structured computer program was developed to provide a research tool for performing

disparity analysis. Entitled program DIDA, this program enables the user to interactively perform certain image operations, select interesting points from a specified image, match these (or other) points to those of another image, and display images (with or without interesting points overwritten) on a CRT. During this program development, three separate image data bases were obtained for processing.

Using program DIDA, the matching efficiency of the Moravec interest operator was evaluated. Also, photographs were taken of the sets of images which were used to test the interest operators. First, a photograph of the original image was taken, then a photograph was taken of the image with the interesting points overwritten. A visual inspection was used to determine if the points were matched for image sets on which matching was performed.

Sequence of Presentation

The sequence of presentation of the thesis material is both logical and functional. Chapter II provides a general description of the DIDA computer program. Sections I and IV of the program are described in detail since they are fairly short. Sections II and III of the program describe the interesting point selection and matching sections respectively, and are only generally described in Chapter II of the report. Chapter III describes the interesting point selection algorithms utilized by program DIDA in detail; whereas Chapter IV describes the interesting point matching algorithm.

The results and conclusions of the effort are detailed in Chapter V. A comparison of the number of interesting points produced versus threshold is given, along with a comparison of the matching efficiency of the Moravec interest operator. Photographs of images from each of the three data bases are shown. Also, photographs of two separate images with the interesting points overwritten are shown. The thesis report concludes with some recommendations for further study contained in Chapter VI.

II. Program DIDA: General Description

As mentioned in the background portion of the introduction section, the DIDA program is to be the third major section of the DIPS program. The overall structure of the DIPS program is shown in block diagram form in Figure 2.

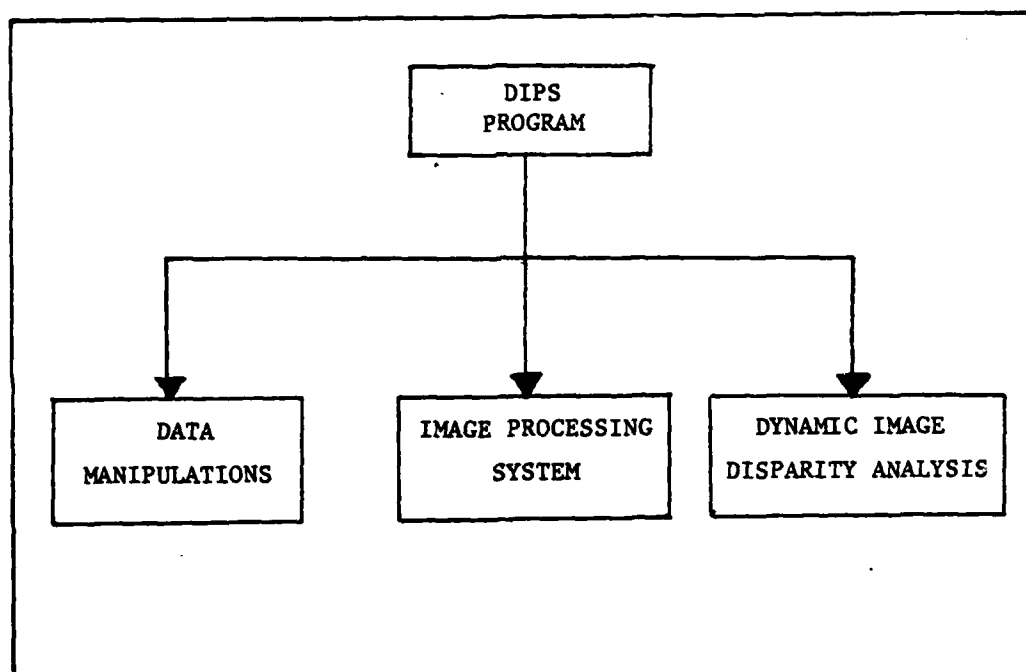


Figure 2. DIPS Program Block Diagram

The data manipulations and image processing system sections are subdivided as shown in Figure 1. The dynamic image disparity analysis section (program DIDA) consists of four major subsections: (1) image operations; (2) interesting point selection; (3) interesting point

matching; and (4) image display. The overall structure of the DIDA program is shown in block diagram form in Figure 3. A glossary of the symbols, variables, and arrays used in the program is shown in Appendix A, along with a brief description of their function. The main control program listing for the DIDA program is shown in Appendix B.

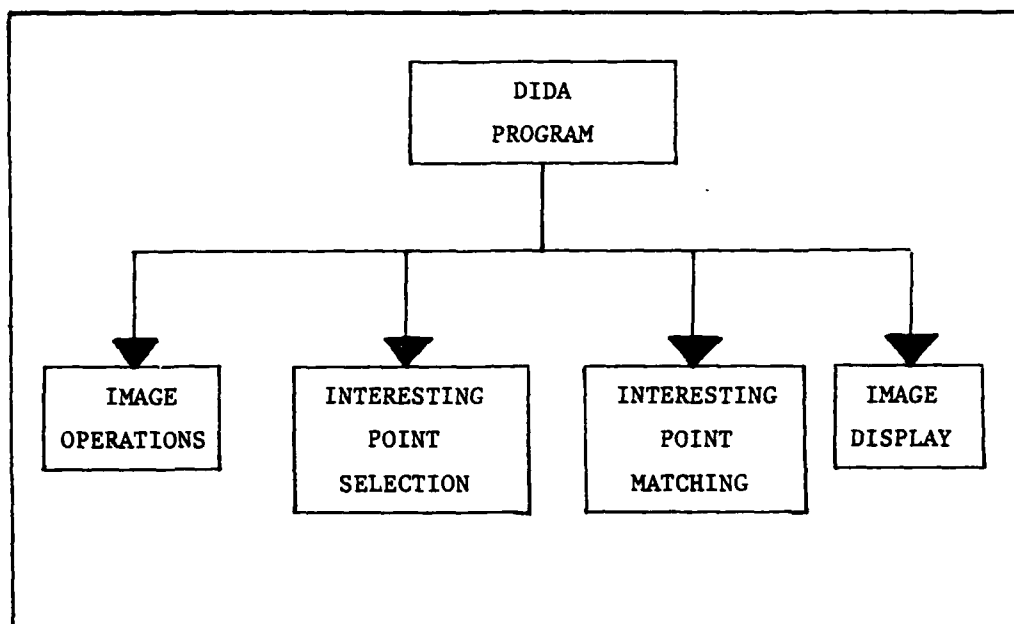


Figure 3. DIDA Program Block Diagram

Program DIDA was developed using a top-down, structured analysis and design technique similar to the one described by Softech [Ref. 7]. Parnas's principle of information hiding [Ref. 8:74,127] was used throughout most of the program to provide a highly reliable and maintainable modular program structure. This was accomplished by using labeled COMMON blocks selectively within each program module to include only information directly relevant to that particular module. In most cases, only error and end program parameters are passed as subroutine

arguments within the program structure.

Program DIDA is an interactive program controlled primarily by user inputs during execution. The specific method of user program control is described during the associated description of the program. In basic terms, the user is asked which program operation (corresponding to one of the four sections mentioned above) is desired. Based on the information input, that operation is then carried out. This interactive method of question/response is used generically throughout the entire program at all levels where a user input is required. Feedback is provided to the user after an information entry is accomplished to enable an entry connection, if desired. Also, user inputs are monitored to insure that nonsensical inputs (such as "YEX" for "YES"), or inputs outside the range of the parameters specified in Table I, are corrected. The operational philosophy, however, is to allow the user to control the program operation. In all cases, the initial assumption is that the user knows the correct input to make. As long as the information input is within the correct parameter bounds, and a typographical error is not committed, the program assumes that the user knows what he or she is doing. A flow diagram depicting the overall operation of program DIDA at the highest level is shown in Figure 4.

Each of the four major sections of program DIDA is now described. Following this, each of the three data bases used for disparity analysis testing is described.

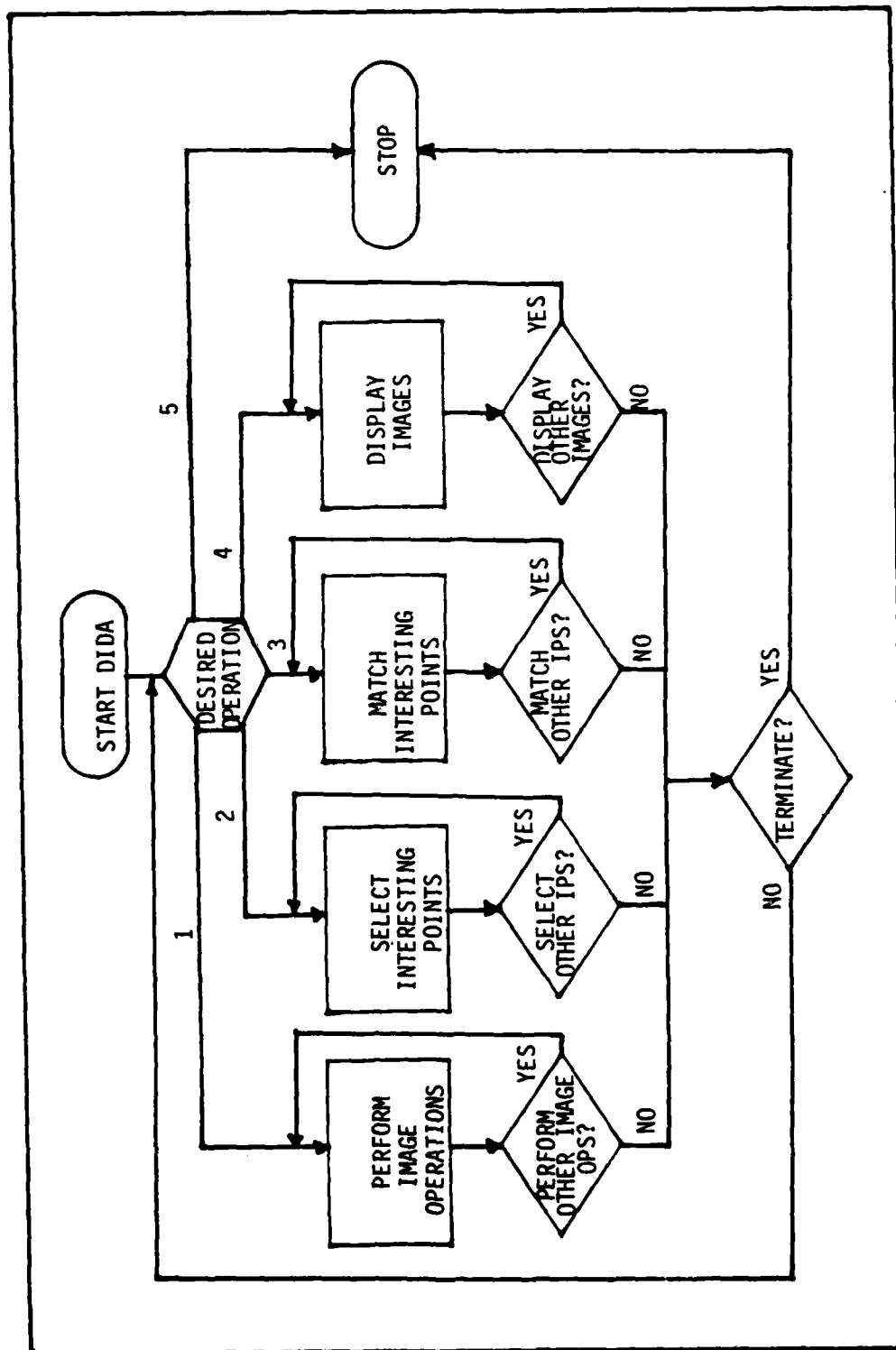


Figure 4. DIDA Program Flow Diagram

Image Operations Section

The subroutine source listings for the image operations section of program DIDA are shown in Appendix C. This section is controlled by subroutine IMGOPR which first allows the user to specify the image size, then select the image operation desired. Three image operations are available: (1) print image intensities on the line printer; (2) delete an image file; and (3) create a new image file. The user may also terminate image operations or program execution, if desired.

Image Size. Images to be processed by program DIDA are stored externally on image files. These files are created from image data bases obtained by the digitization of photographs taken of real-world scenes, or by creating them using one of program DIDA's image creation algorithms. Each image is represented by an array of numbers, where each number within the array (array element) is a binary representation of the intensity of the elemental area of the real world scene which maps to that location. The binary representation of each intensity array element (called a "pixel" - for picture element) is an 8-bit field containing binary numbers from 0 (totally dark) to $2^8 - 1 = 255$ (totally white). Thus, the range of intensities entering the digitizer from the image photograph is scaled to this 8-bit range of numbers. In other words, each image pixel is assigned a number between 0 and 255 by the image digitizer which is proportional to its actual intensity. These pixel values are then sequentially output to a magnetic tape or storage disk as the image is scanned from pixel to pixel. This image

digitization process is shown in Figure 5.

The size of an image is defined as the total number of pixels it contains. As mentioned above, these pixels are stored in an intensity array. This array is specified by the number of rows and columns it contains, and the product of these rows and columns must equal the image size. When an image is digitized, the number of columns it contains is determined by the number of pixels required to span its horizontal length. The number of rows is determined by the number of pixels required to span its vertical width. When the digitized image is stored on an image file, it is stored sequentially from pixel-to-pixel, and from row-to-row. As a matter of convention, one complete row of pixels is usually taken to form one logical file record. Thus, the number of records in the image file is equal to the number of rows of the image, and the number of data fields within each record is equal to the number of image columns. Therefore, the product of the number of image file records and data fields within each record defines the image size for that file. Consequently, the number of rows and columns of the intensity element storage array of program DIDA must correspond directly with the number of rows and columns (records and record fields) of the image file.

The image size is specified through subroutine IMGSZE. This subroutine requests that the user input the number of rows and columns of the image. It then calls subroutining IRCFND to obtain this information, and prints it on the line printer. The image size and its

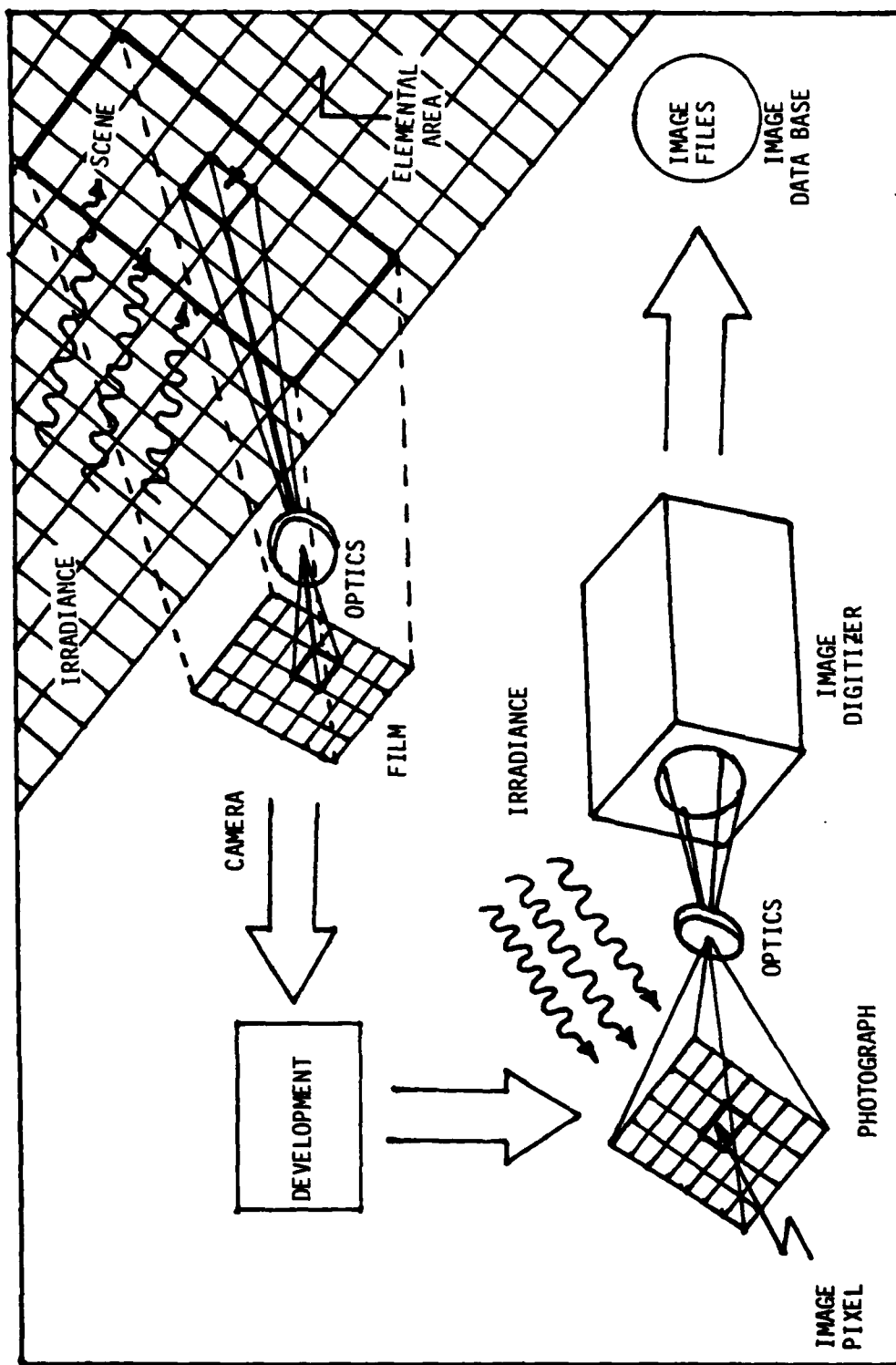


Figure 5. Image Digitization Process

inverse are also printed.

Subroutine IRCFND calls subroutines IMROW and IMCOL to obtain the number of image rows and columns respectively. If the user feels that an error has been committed, an affirmative response may be offered when queried for error checking. If the user requests an error check, subroutine IRCCHK is called. This subroutine displays the input information and asks the user if it is correct. If a negative response is entered, subroutine IRCCOR is called to allow the user to correct the erroneous input. A flow diagram illustrating the method of inputting the image row and column size is shown in Figure 6.

The information obtained after calling IMGSIZE is stored in the area COMMON /IMAGE/ for later use. The variables IMGROW, IMGCOL, IMGSIZE, and IMGINV specify the values obtained. Note that in each subroutine, only information pertinent to the function performed by that subroutine is present. This exemplifies the principle of information hiding mentioned earlier. An illustration of the manner in which the image is described by program DIDA is shown in Figure 7. The values of L, M, and N are described in Table I.

After the image size has been specified, subroutine IMGOPR calls subroutine IMOSL to allow the user to select the image operation desired. IMOSL lists the options available and the user enters the number corresponding to the one to be performed.

Printing Image Intensities. If the user enters number one in

IMOSEL, the image operation required to print the intensity values of a specific image file on the line printer is selected (subroutine IMOPRI).

This subroutine first requests that the user enter the logical unit number and file name of the image file to be printed. Subroutine FLUFND is then called to obtain this information from the user. This subroutine calls subroutines LUNBR and FILENM to obtain the logical unit number and file name respectively. As in the case of the image size, the user is asked if entry checking is desired. If so, subroutine FLUCHK is called. Error correction is then accomplished by subroutine FLUCOR, if needed. This file information is stored in the area COMMON /FILE/ and specified by the variables LUN and FNAME. FNAME is a 34 element byte array which stores the alphanumeric literal specifying the file name. A flow diagram illustrating the method of inputting the logical unit number and file name is shown in Figure 8.

Once the logical unit number and file name have been specified, IMOPRI calls subroutine OPNOLD which opens an unformatted, direct access, sequential file which was previously created (i.e., already in existence). The variables IMGROW and IMGCOL previously specified are used to determine the number of records and record length respectively.

Subroutine INPUT is called after the file is opened. This subroutine reads the logical record from the image file which corresponds to the value of RECNUM and stores it in one row of array IOBUFR. This array is located in the area COMMON /BUF/.

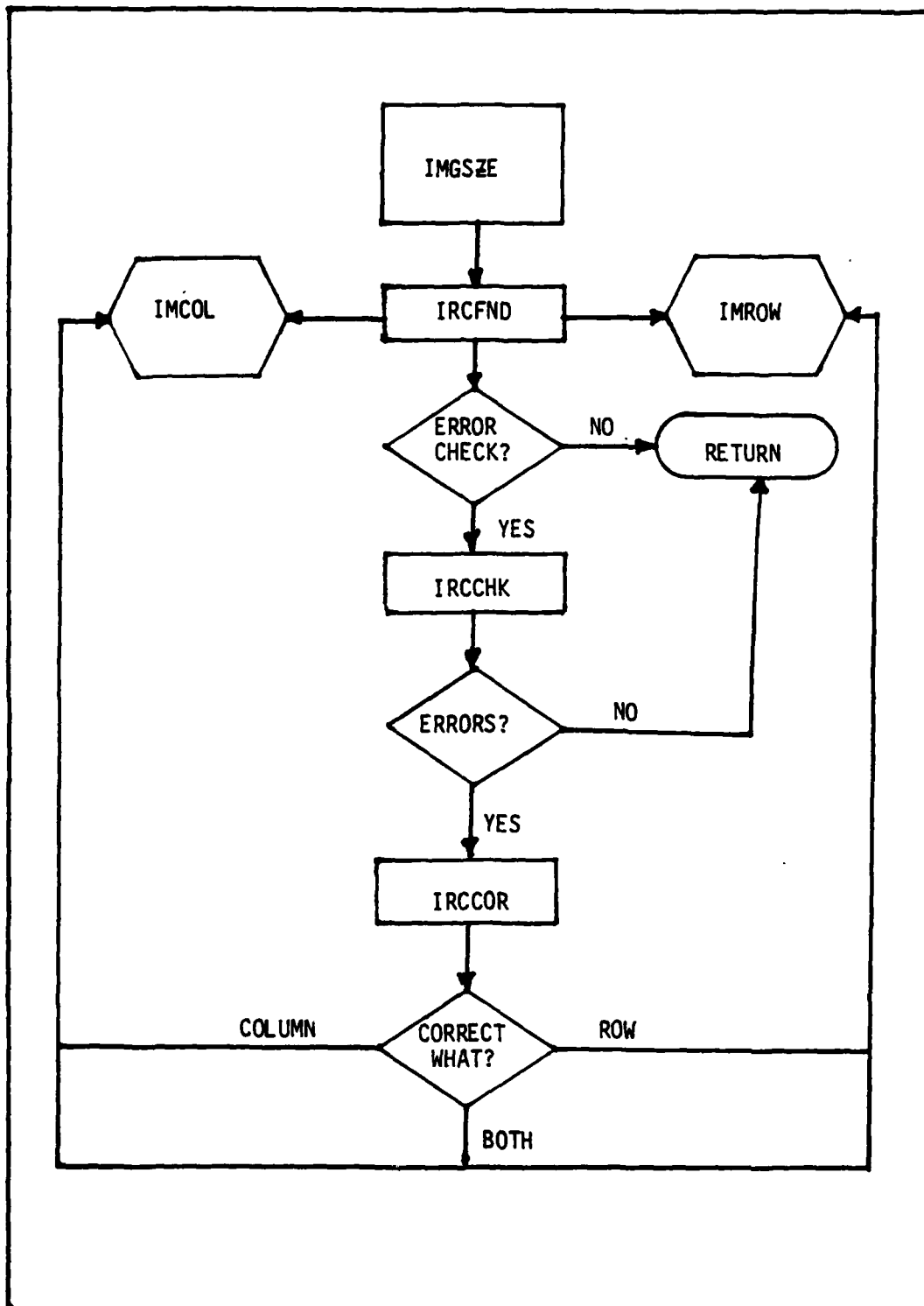


Figure 6. Image Row and Column Size Input Flow Diagram

IMGSZE = IMGROW X IMGCOL

IMGINV = 1/IMGSZE

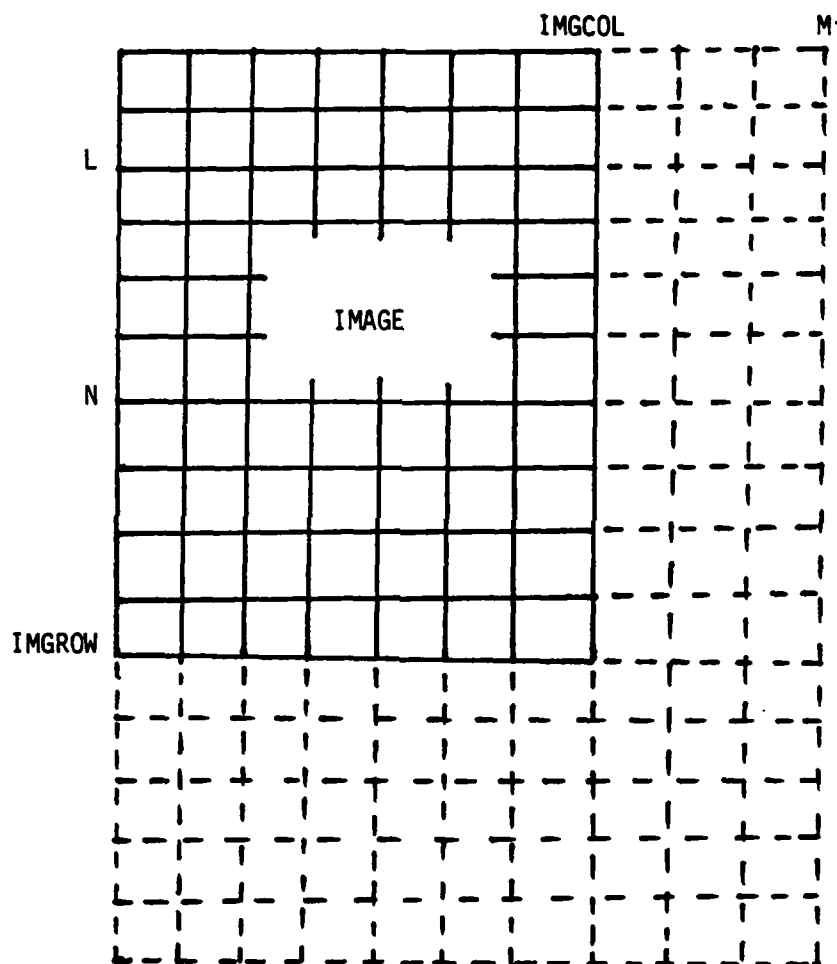


Figure 7. DIDA Program Images Description

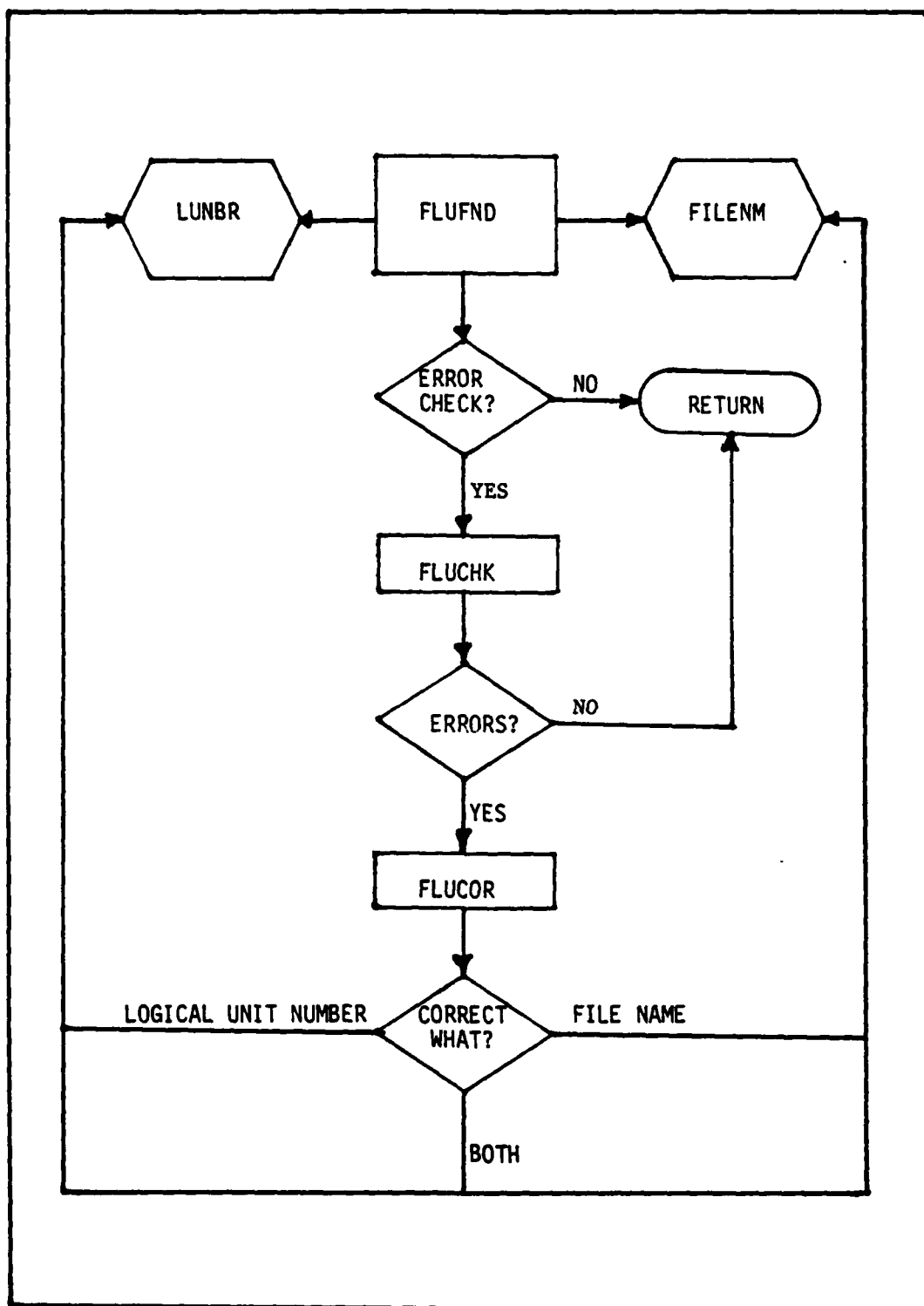


Figure 8. Logical Unit Number and File Name Input
Flow Diagram

In order to print the intensity data, it is necessary to convert the 8-bit intensity fields stored on each logical record to a 16-bit data word to be compatible with the format accepted by the line printer. It could be assumed that all that is necessary to accomplish this conversion is to merely establish equivalence between the 8-bit array and a 16-bit array. Unfortunately, the conversion process is not quite that simple. The problem results from the form the data assumes when it is stored on the image file. Recall that each record field is an 8-bit number between 0 and 255. When the computer reads an 8-bit number, it assumes that it may be either positive or negative in the range from -128 to +127. If an 8-bit number greater than 127 is input, the computer treats this as a negative number because it changes the sign bit of the byte memory storage location in which the 8-bit number is stored. Thus, it is necessary to convert the 8-bit intensity value to the correct 16-bit data word if the number input is greater than 127. This byte-to-word number conversion procedure is explained in Appendix G.

After this byte-to-word number conversion process has been accomplished, the data is output to the line printer and printed. Array INTBUF is the 16-bit array used to store the converted 8-bit data in the area COMMON /INBUF/. Upon completion of data printing, the image file is closed using subroutine CLOSKP. This subroutine closes the file with the DISPOSE = 'KEEP' option specified so that it will not be deleted.

File Deletion. If the user enters the number two in IMOSL, the image operation required to delete an existing image file is selected (subroutine IMOPR2). The logical unit number and filename are specified by subroutine FLUFND, and subroutine OPNOLD opens the file. After this, subroutine CLOSDL is called with the DISPOSE = 'DELETE' option specified. This deletes the file.

Image Creation. If the user enters the number three in IMOSL, the image operation required to create a new image file is selected (subroutine IMOPR3). Again, the logical unit number and file name are specified by subroutine FLUFND interactively. However, this time subroutine OPNNEW rather than OPNOLD is called. OPNNEW opens a new unformatted, direct access, sequential file whose record size and number of records are specified by the values of IMGCOL and IMGROW respectively. Subroutine OPNNEW may also open an existing file for modification, if desired.

After opening the file, subroutine IMASEL is called. This subroutine allows the user to select the algorithm (method) to be used to find the intensity values of the pixels of the new image to be created. Three algorithms are available: (1) random intensity distribution; (2) user terminal input; and (3) card reader input. The user may also terminate program execution, if desired. When the new image has been created, subroutine IMOPR3 closes the image file using subroutine CLOSKP. Subroutines IMOPR4 and IMOPR5 are not used at the present time.

Random Intensity Distribution. If the user enters the number one in IMASEL, the image creation algorithm required to create an image with a random intensity distribution is selected (subroutine IMALG1). This subroutine assigns randomly selected values between 0 and 255 to the image elements of the image being created. This is accomplished using the FORTRAN subprogram RAN(SEED), where SEED is an INTEGER*4 seeding number for the random number generator.

Subroutine IMALG1 first calls subroutine SEEDFD to allow the user to interactively specify the desired value of SEED. Again error checking is performed. Each different value of SEED produces a new random number sequence. Since the random number subprogram produces a number between 0 and 1, multiplying its output by 256 will produce the necessary intensity range. Since only integer numbers are used, and "1" is never output by RAN, the largest number which can result is 255. Also, numbers less than 1 will be truncated to "0" (zero). Thus, the desired range is obtained.

After each random intensity value is generated, it is stored in array INTBUF. This value is then copied into array IOBUFR for output to the image file. When an entire row of intensity values has been stored in one row of IOBUFR, subroutine OUTPUT is called. This subroutine outputs the row of intensity data to the image file record specified by the value of RECNUM. If the debug program version is being executed (see Chapter V), these values are also printed on the line printer.

User Terminal Input. If the user enters the number two in IMASEL, the image creation algorithm required to create an image using the user's terminal is selected (subroutine IMALG2). This subroutine allows the user to interactively enter the intensity values corresponding to the pixels of the new image being created. The user is asked to enter the intensity values one record at a time, and is then given information on how to enter the numbers. Each record is terminated with a slash (/), and a carriage return (CR). The user - specified intensity values are then stored in one row of array IOBUFR and written to the image file with subroutine OUTPUT. They are also printed on the printer if the debug program is used.

Card Reader Input. If the user enters the number three in IMASEL, the image creation algorithm required to create an image using the card reader is selected (subroutine IMALG3). This subroutine allows the user to input the intensity values corresponding to the pixels of the new image being created using cards. Each card contains several intensity values. When the computer encounters a slash, an end-of-record is indicated. This record is then stored in one row of array IOBUFR, and then written to the image file using subroutine OUTPUT. If an error occurs during card reader entry, the file is closed using subroutine CLOSDL. The input intensity values are also printed on the printer if the debug program is used. To perform image operations, then, it is necessary to just define the image size. Next, the desired operation is selected. Finally, if image creation is desired, the image creation algorithm to be used is selected.

Subroutines IMALG4 and IMALG5 are not used at the present time. An overall flow diagram of the image operations section of program DIDA is shown in Figure 9. Refer to Figure 6 and 8 for a more detailed diagram of the image size and file description processes.

Interesting Point Selection Section

The subroutine source listings for the interesting point selection section of program DIDA are shown in Appendix D. This section is controlled by subroutine INTPTS. As in the image operations section, the image size is defined first. Next, the size of the processing window to be used to select the interesting points is defined. After this, several rows of the image to be processed are input depending upon the window size. The user is next asked if the interesting points are to be stored on an output file. If an affirmative response is offered, the correlation window size is defined, and a scratch file for storing the correlation windows surrounding each interesting point selected is opened. If the user does not wish to store the interesting points, these steps are skipped.

Once these initial "set-up" operations have been performed, one of three interest operators is selected to find the interesting points on the selected image. These interest operators are: (1) simple variance; (2) directed variance; and (3) edged variance. The interesting points of the image are then found using the selected operator. If the user indicated that the interesting points should be stored, a logical unit number and file name are specified, and the

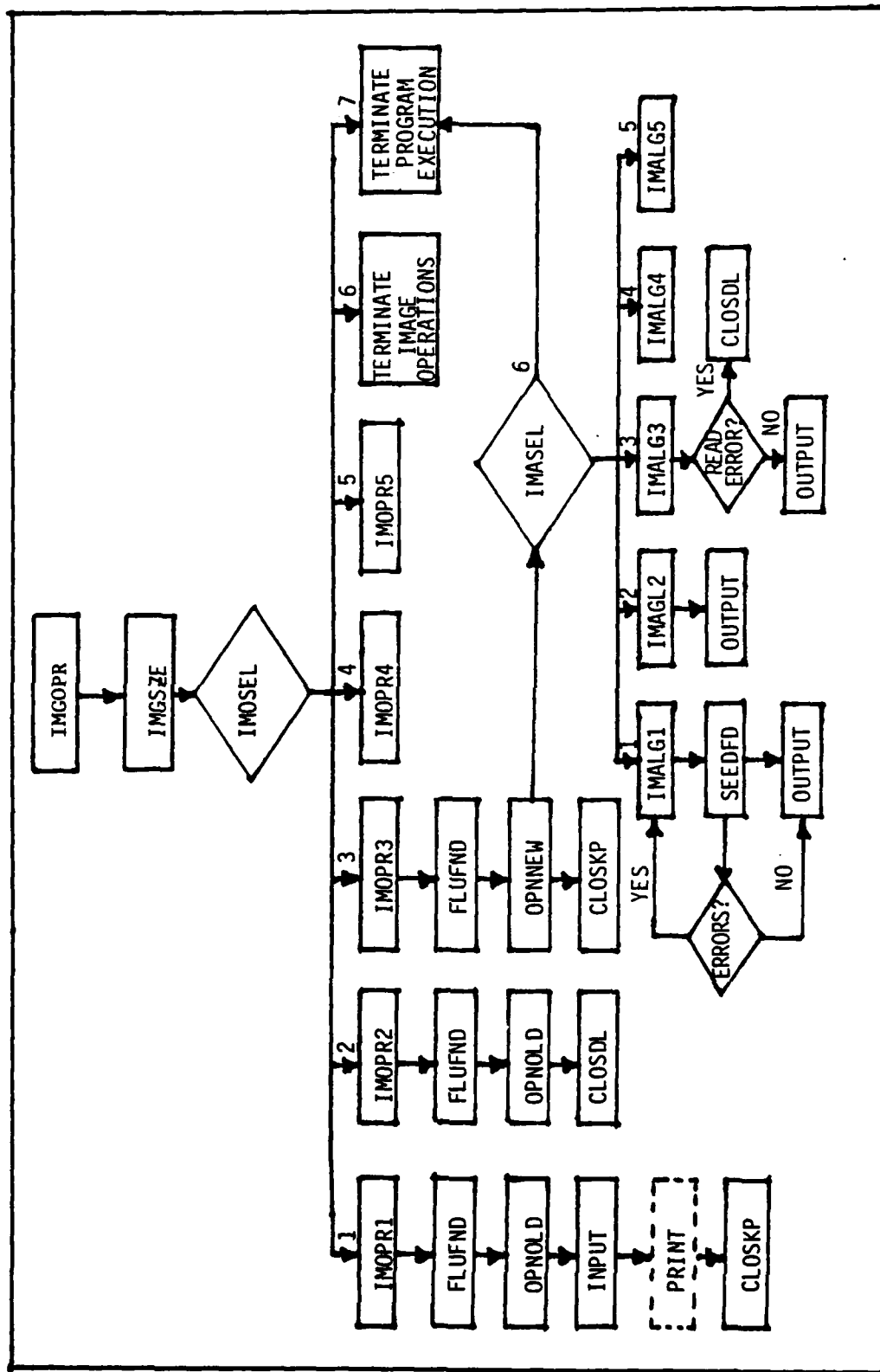


Figure 9. DIDA Image Operations Section Flow Diagram

output file for storing them is opened. The interesting points, along with their corresponding correlation windows (used for matching) and other associated information, are then written to the file. If the user does not specify interesting point storage, these steps are skipped.

Finally, the user is asked if the interesting points should be listed on the line printer. If an affirmative response is entered, the points are listed; otherwise, they are not. Thus, the user has the options of either storing the interesting points on an output file, listing them on the line printer, or both (or neither).

In the following sub-sections, each of the major operations associated with the interesting point selection section is discussed. As mentioned in the sequence of presentation section of the introduction, the interesting point selection algorithms are only generally described in this section. These algorithms are described in detail in Chapter III. Also, the definition of the image size was discussed in the image operation section above, and will not be repeated here.

Window Size. In order to select interesting points on a given image, a window of various sizes may be passed over the image to "look at" small sections of the image one at a time. If an interesting feature of the image is found within this section, an interesting point may be defined to exist there. Thus, before interesting point processing can proceed, it is necessary to determine the size of the

window to be passed over the image of interest.

Specification of the window size for processing is accomplished in exactly the same manner as the specification of the image size. Subroutine WNDSE calls subroutine WRCFND to obtain the number of window rows and columns. Subroutine WRCFND calls subroutines WDRW and WDCOL to permit the user to interactively enter the number of window rows and columns respectively. Subroutines WRCCHK and WRCCOR are used for entry error checking and correction, if needed. The flow diagram for inputting the window row and column size is identical to the one shown in Figure 6.

The information obtained after calling WNDSE is stored in the area COMMON /WINDOW/ for later use. The variables WNDROW, WNDCOL, WNDSE, and WNDINV specify exactly the same information as the corresponding image variables. Two new variables are associated with the window that were not used for the image. These are WNDRWH and WNDCLH. These variables contain the values WNDROW/2 and WNDCOL/2 respectively, and are used to locate the center of the window. These variables are also stored in the area COMMON /WINDOW/. An illustration of the manner in which the window is described by program DIDA is shown in Figure 10. The values of M and N are described in Table I.

The maximum value that WNDCOL can take on (i.e., that is permitted to be entered by subroutine WDCOL) is equal to the number of columns input for the image (IMGCOL) minus one. Since the maximum value that IMGCOL can take on is M, then the maximum value that WNDCOL can take on

when IMGCOL is at its maximum value is $M-1$. Similarly, the maximum number of rows which may be input before the intensity array storage space is exceeded is $2N$. The maximum value that WNDROW can take on is restricted by subroutine WDROW to $2N-1$. The reason that one is subtracted from the absolute maximums in both cases is that it is assumed that when the user specified a window size, a processing operation is desired. This implies window movement. Reducing the window row and column size by one allows for movement of one pixel in each direction.

Image Input. Ideally, it would be efficacious to be able to input an entire image, regardless of its size, into an intensity storage array within program DIDA during execution. However, due to the memory storage size restrictions imposed by the computational system being used (a DEC PDP 11/45 minicomputer), this is not possible for large images. The maximum amount of memory available for program storage during execution (including other system overhead storage requirements such as error checking programs) is 32K (32768) storage locations. Each location contains a 16-bit data word, and each of these words contains 2 8-bit bytes. Thus, for rather large programs (such as program DIDA), the amount of array storage space is severely limited due to the program size.

The memory storage problem can be alleviated somewhat in two ways:
(1) divide the program up into segments, and use virtual addressing to overlay portions of the program, and (2) use a VIRTUAL storage array to

WINDINV = 1/WNDSZE

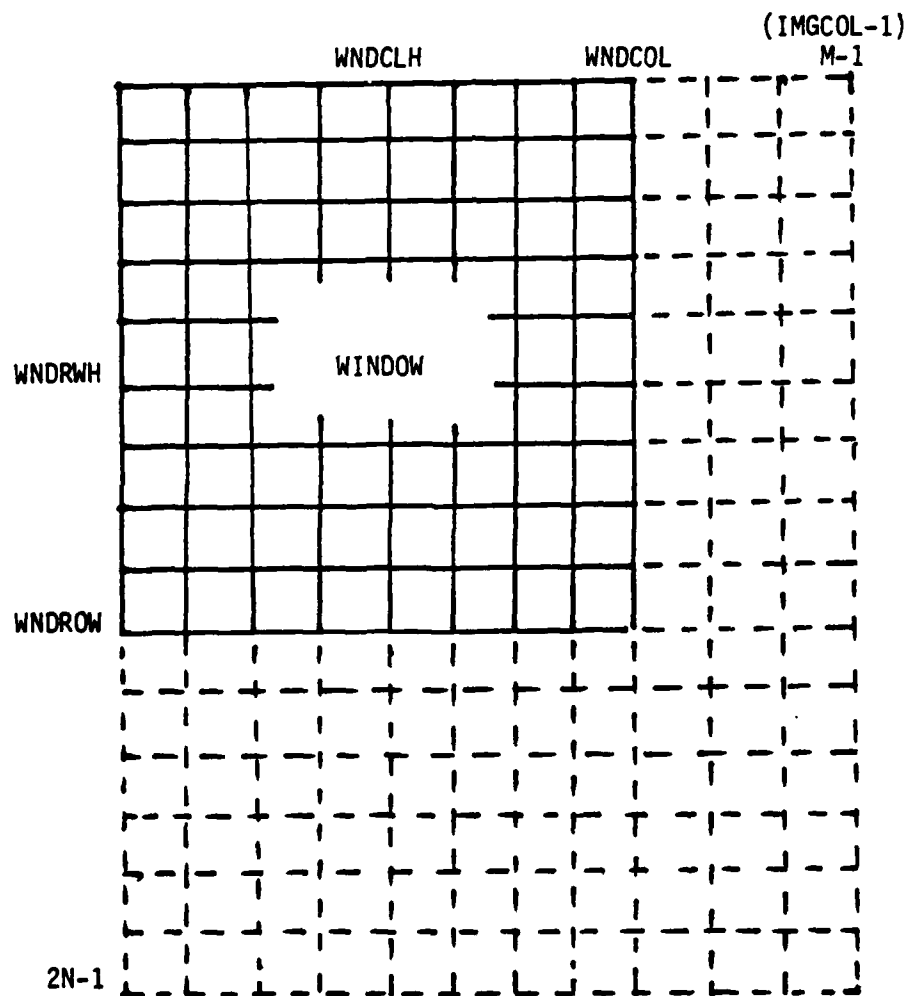


Figure 10. DIDA Program Window Description

map outside the memory location in which the program is running to another 32K memory storage area which may be totally used for storing intensity values in an intensity array (or any other information). To accomplish program overlays (see DEC Manual for a more detailed discussion), it is first necessary to divide the program up into segments which are functionally independent of each other. A root segment (DIDA control program) is left in core (bipolar memory) at all times during program execution. This Root program contains information common to all the overlay segments (e.g., the COMMON block data). As program execution proceeds, these segments are brought into core from their storage location on the system disk, under operating system control (RSX-11M), in accordance with the overlay structure specified during the building of the task program. When a certain segment has performed its function (finished executing), it is taken out of core and another segment is brought in and executed. This process continues until the execution of the entire DIDA program is terminated by the user. Since the DIDA program was developed as a top-down structured program, it readily lends itself to being broken up into overlay segments which are similar to the program structure.

Even though a great savings in memory storage space is realized using overlays, it is still not sufficient to provide enough array space for image storage. To obtain even more storage space, a VIRTUAL array is used. Since a machine with a 16-bit wordlength is being used, the largest amount of memory space which can be mapped is $64K (2^{16} = 65536)$ 8-bit bytes. Since 16-bit words are being used, this means 32K

words are available for the program (as mentioned above). However, several of these 32K storage locations exist within the overall core memory structure of the computer to provide for multiuser operation. This means that simultaneous tasks may be executed on the system by the operating system up to the number of 32K memory sections available. It also means that these other 32K sections can be made available for use by a task executing in a different 32K section by using a VIRTUAL array. When a VIRTUAL array is specified within a program task, this flags the operating system to specify a 4K register in the task program (called a window) to be used to map outside the task program to another 32K area. This storage space is totally available to the task program. Thus, at the expense of 4K of space within the task program, another 32K of array storage space is obtained. The drawback, of course, is slower execution time (not a big factor since a real-time program was not the aim of this particular study). The structure of the overlay and VIRTUAL array storage scheme is shown symbolically in Figure 11.

For extremely large images, however, even overlaying the program and using a VIRTUAL array is not sufficient to provide enough storage space to contain the entire image. For example, for a 512X512 image (the largest considered for program DIDA), 262144 words of array space would be required to store the entire image. Since this much storage space is not available, it is necessary to buffer the image into the program in sections as it is processed. Recall that for processing, a window is moved across the image from column-to-column and row-to-row. Thus, processing is accomplished by scanning the image from

left-to-right, and from top-to-bottom with the window. After the window has made one complete sweep across the image horizontally (from left-to-right), it is moved back to the left edge of the image again and moved down one row from its previous position. It is then again swept across the image horizontally, moved back to the left, and moved down another row. This process continues until the window reaches the lower right hand corner of the image. This window movement process is shown pictorially in Figure 12 (a) thru (i).

Once the window has made one complete horizontal sweep across the image and moves down one row, the row that it leaves when it moves down is no longer needed. Therefore, the method of image input decided upon is to replace each row of image data below the window as processing progresses. In other words, since the image cannot remain static in memory while the window moves across it, the window is passed across the image from left-to-right, then moved back to the left again. The uppermost row of image data just processed is replaced with the next row just below the window. So, in effect, the window moves back and forth in a stationary path while rows of data within it are replaced as they are processed in a row rotation fashion.

To explain this process in more detail, refer to the illustrations of the method of image input during processing shown in Figure 13 (a) thru (o). Figure 13(a) shows the configuration of the system at the beginning of image processing. This corresponds to the window position shown in Figure 12(a). The portion of the image within the area

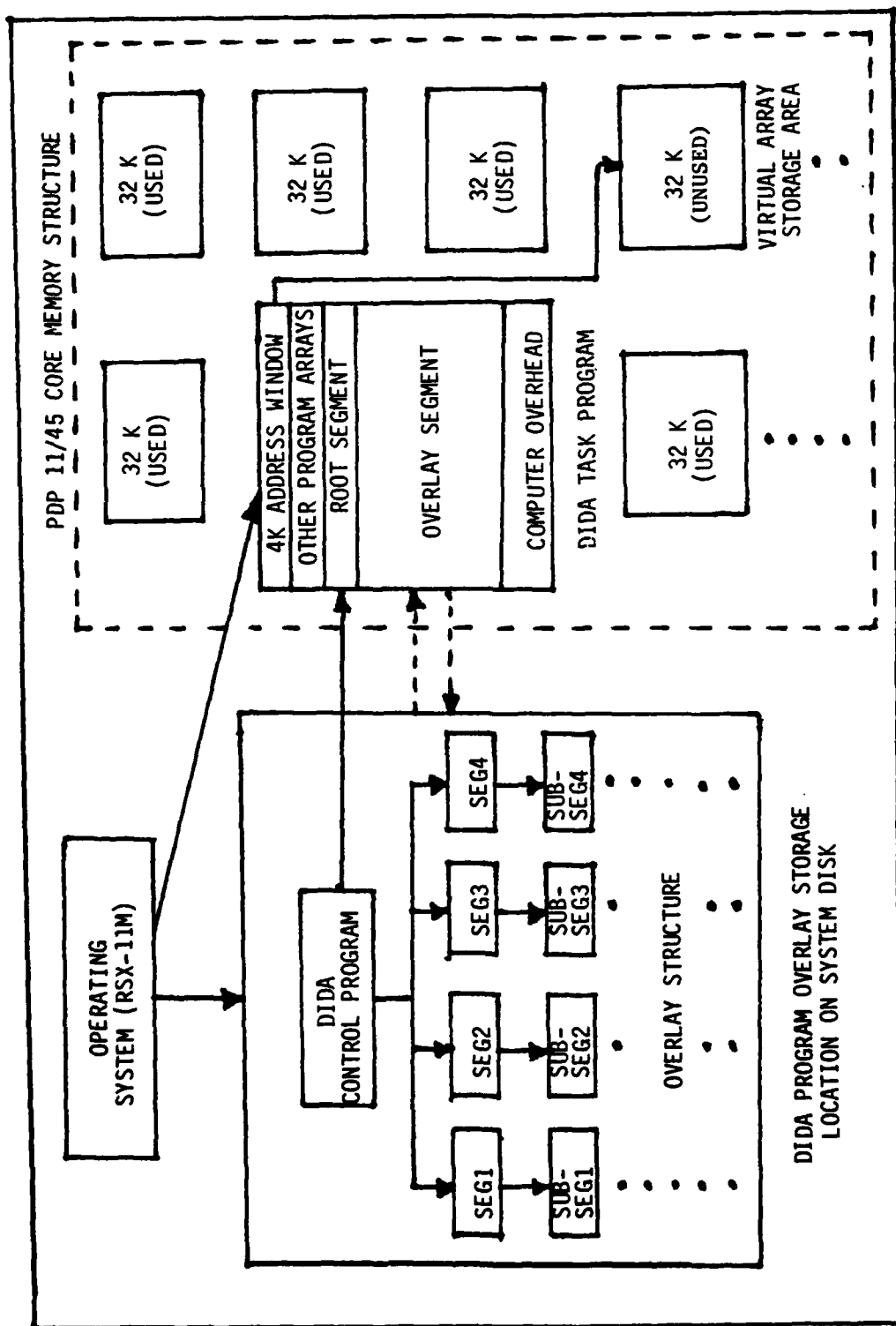


Figure 11. DIDA Program Overlay and Virtual Array Structure.

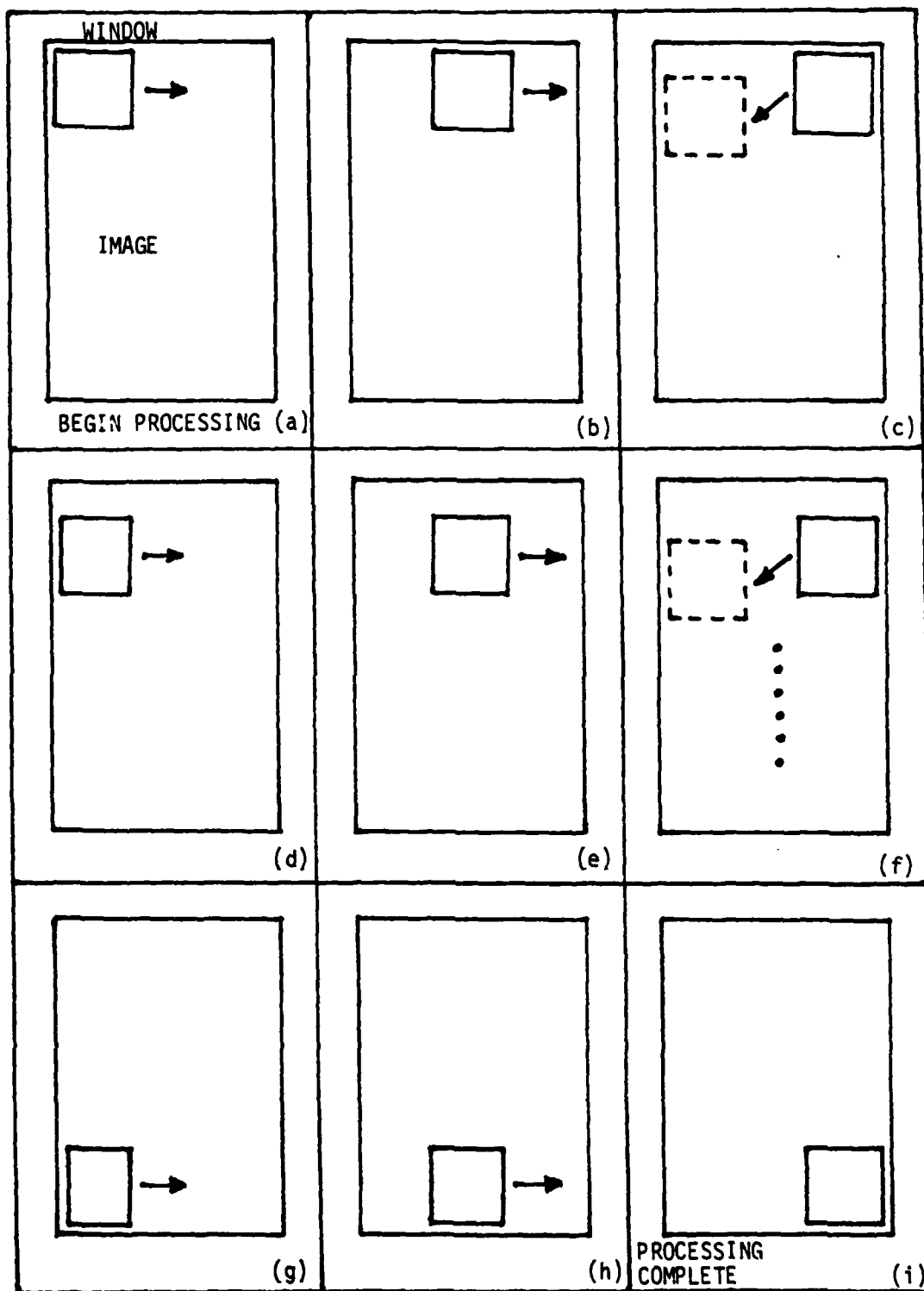


Figure 12. Window Movement During Processing

covered by the window in the upper left corner of the image is now processed. The window is advanced horizontally one column position and that corresponding window area is processed. This window movement continues until the window reaches its rightmost position on the image as illustrated in Figure 12 (b) and (c). After the rightmost window position is processed, the window should move back to the left edge of the window and down one row as shown in Figure 12(d). In actuality, rather than moving the window down one row, the first row of the intensity array across which the window moves is replaced with the first row of the input buffer array as shown in Figure 13(b). Since this row was processed on the first pass of the window across the array, it is no longer needed for processing. The first row of the input buffer corresponds to the row of the image which was just below the window when the window was at the top of the image. Thus, replacing the first row of the intensity array with the first row of the input buffer has the same effect as actually moving the window down one row on the image. After this row change operation has taken place, the window is again moved across the intensity array horizontally as processing continues as shown in Figure 12 (e) and (f). Although the sequence of window movements shown in Figure 12 is not actually taking place in the manner portrayed, it is convenient to think of the movement as taking place in this fashion to avoid confusion.

As processing continues, successive rows of data within the intensity array are replaced with rows of data from the input buffer as shown in Figure 13 (c), (d), and (e). When the last row of data

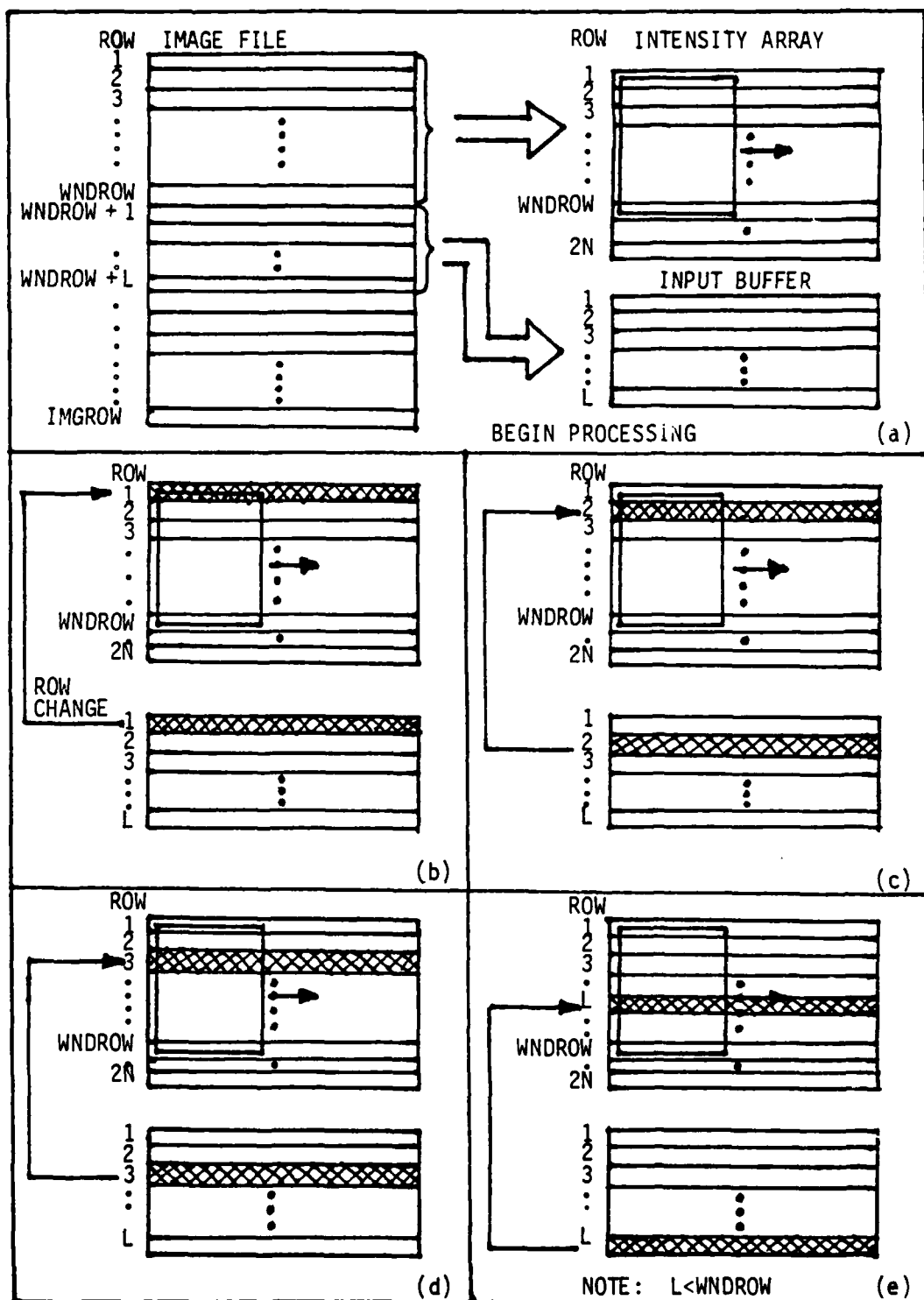


Figure 13. Method of Image Input During Processing

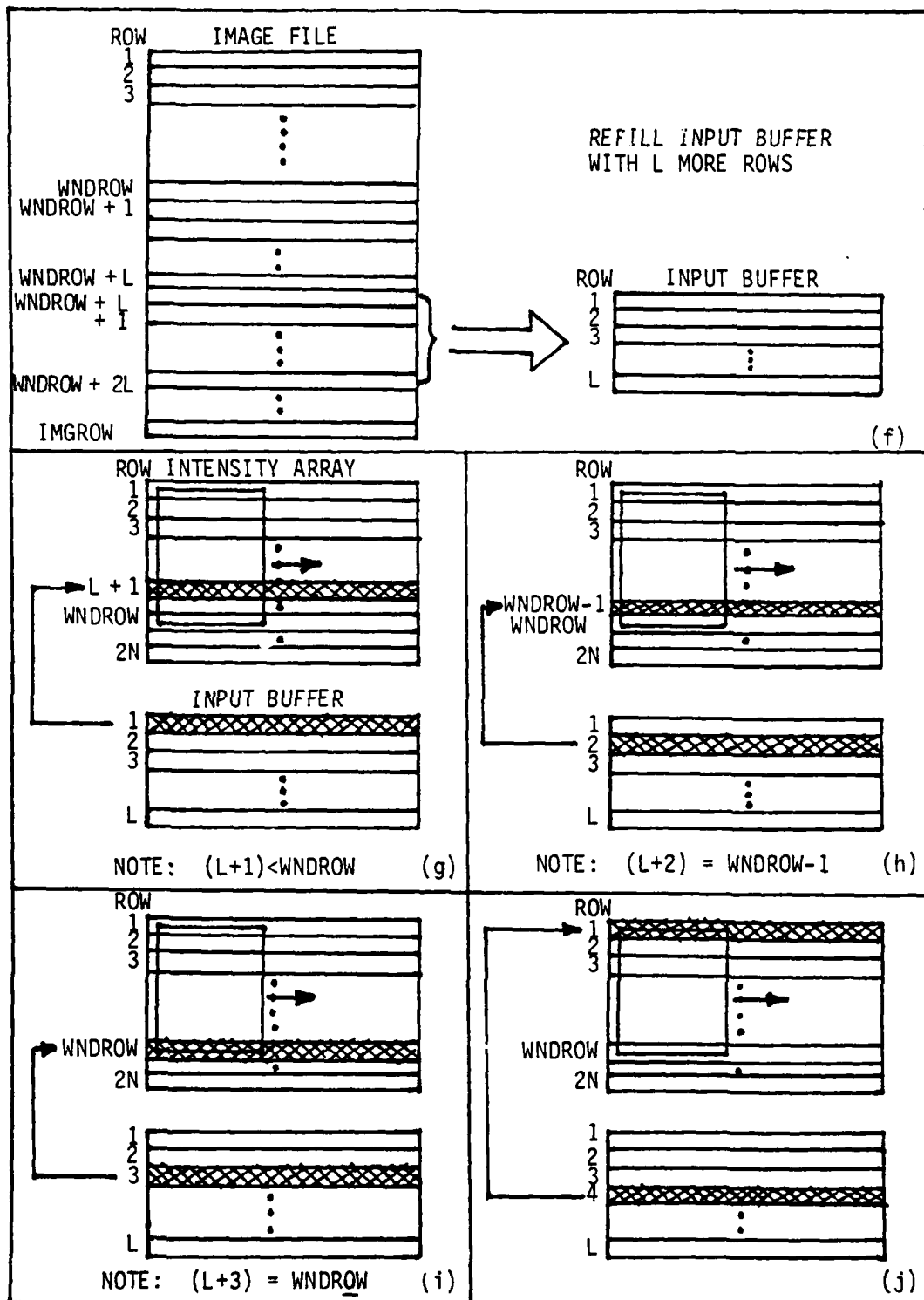


Figure 13. Method of Image Input During Processing

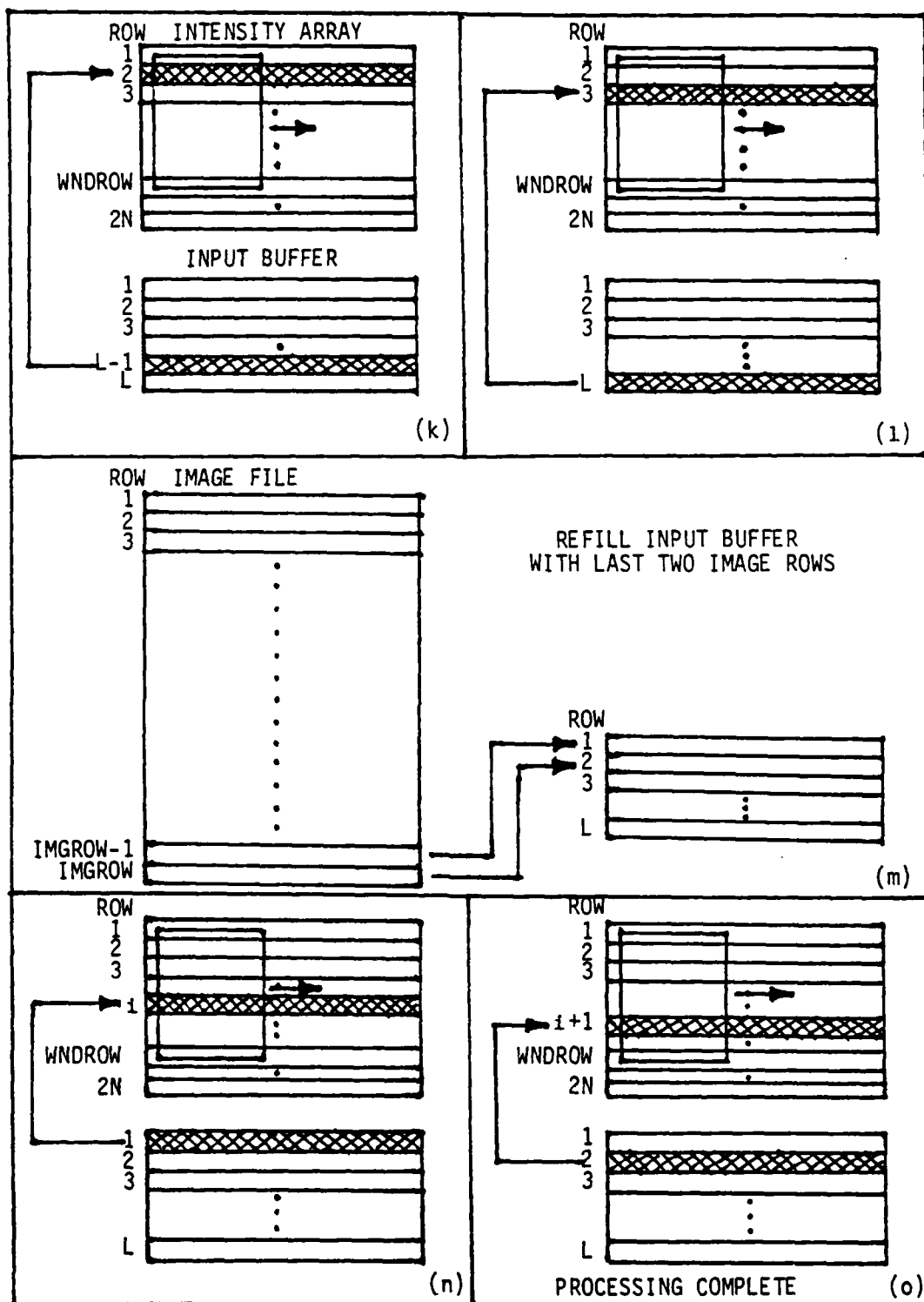


Figure 13. Method of Image Input During Processing

from the input buffer has been used, this buffer is refilled with L (see Table I) more rows of data from the image file (assuming the end of the image is not reached) as shown in Figure 13(f). Processing then continues as it did before the buffer was refilled. Now, however, rows of data are taken starting from the first buffer row again as shown in Figure 13 (g) thru (l). Note that after the input buffer is refilled the first time, there is no longer a one-to-one correspondence between the input buffer rows and the intensity array rows being replaced. Note also in Figure 13 (i) and (j) that after the last window row of data has been replaced, the row change operation begins at the top row of the intensity array once again.

Finally, after enough rows of data have been input from the image file, the next time the input buffer is refilled the last image row will be input. This is shown for the last two image rows (arbitrary choice) in Figure 13 (l) and (m). The window is then passed across the intensity array two more times to complete the processing as shown in Figure 13 (n) and (o). Figure 13(o) corresponds to the last pass of the window across the image as shown in Figure 12 (g), (h), and (i). Thus, as the window is moved back and forth across the intensity array, rows of data within the array are replaced with rows of data from the input buffer in a row rotation fashion until the last image row has been processed.

In program DIDA, the image input process is begun after the image and processing window sizes have been specified by calling subroutine

IMGIN. This subroutine first calls subroutine FLUFND to allow the user to specify the logical unit number and file name of the image file to be processed, as shown in Figure 8. Next, subroutine OPNOLD is called to open the file for input. After this, several rows of image data are input into the VIRTUAL array INT (beginning with the first row) by calling subroutine INPUT. The number of image rows input is equal to the value of WNDROW. If the debug program is being executed, these data rows are also printed on the line printer.

After the intensity array has been initialized, the next L rows of image data succeeding the ones input into INT are read into the input buffer IOBUFR by calling subroutine BUFRIN. The operation of subroutine BUFRIN is explained in more detail in a later section. Program DIDA is now ready to begin processing with the system state as shown in Figure 13(a).

The dimension of the VIRTUAL intensity storage array INT is $2N \times M$. The reason that $2N$ is used is because INT is an array of integer numbers. In program DIDA's main control program (also called PROGRAM DIDA), a generic VIRTUAL storage array STORE of dimension $N \times M$ is declared. This array is a REAL*4 array in the main program. It is declared a REAL*4 array because it is also used in the interesting point matching section of program DIDA to store the labels and probabilities (and other information) associated with the matching algorithm. Since some of this data is REAL*4, the array must be declared REAL*4 in the main program to allocate that much VIRTUAL array

space (Figure 11). Each REAL*4 variable contains two 16-bit data words. Thus, if 32K words may be mapped in a VIRTUAL array, then only 16K (16384) REAL*4 values may be stored. Since the largest value that IMGCOL will take on for any given image to be processed is 512, the maximum value that N can take on is $16384/512 = 32$.

When the VIRTUAL storage array STORE is passed to the interesting point selection section of program DIDA from the main program, it is then only used to store integer values in the range from 0 to 255. Each intensity value requires only one 16-bit word for storage after it has been converted from its 8-bit representation on the image file (see Appendix G). Since only one 16-bit word is required for each array location, 32K (32768) VIRTUAL array locations are available for storage of the intensity values. The maximum image column size M is set to the largest value that IMGCOL will take on for any image to be processed (512). This leaves the maximum number of rows available as $32768/512 = 64$. However, N is assigned a value of 32 in the main program. Thus, the VIRTUAL array INT is dimensioned 2N X M in the interesting point selection sections.

Interesting Point Selection. After the image and processing window sizes have been described, and the intensity array and input buffer have been initialized with image data, program DIDA asks the user if the storage of the selected interesting points is desired. If a negative response is given, interesting point processing begins. If the user requests interesting point storage (affirmative response), it

is assumed that the points are being stored for a later matching operation (Chapter IV). One of the major features of the matching algorithm is assigning initial probabilities to the labels associated with each candidate match point. These probabilities are then iteratively updated to determine the points which match. To assign the initial probabilities for matching, a window is formed around each interesting point selected for both sets of interesting points being matched. For points in each set which are in close proximity to one another, their corresponding windows are compared (correlated) in one of a number of ways (the exact correlation method to be used is specified by the user during program execution). The initial probabilities for matching are assigned based on the correlation of the two windows compared.

If the user has specified interesting point storage, as interesting points are located during processing, their correlation windows are stored on a temporary "scratch" file for later retrieval. This is done because there is not sufficient storage space in core to store all the windows for all the interesting points. The size of the correlation window to be placed around each interesting point when located is specified by the user in subroutine CWDSZE. This subroutine is called by INTPTS after the user requests interesting point storage. Specification of the correlation window size is accomplished in exactly the same manner as the specification of the processing window size for interesting point selection. Subroutine CWDSZE calls subroutine CRCFND to obtain the number of correlation window rows and columns. CRCFND

then calls CWCROW and CWCOL to permit the user to interactively enter the number of correlation window rows and columns respectively. Subroutines CRCCHK and CRCCOR are used for entry error checking and correction, if needed. The flow diagram for inputting the correlation window row and column size is identical to the one shown in Figure 6.

The information obtained after calling CWDSIZE is stored in the area COMMON /CORWND/ for later use. The variables CWDROW, CWDCOL, CWDROWH, CWDCLH, CWDSIZE, and CWDINV specify exactly the same information as the corresponding processing window variables. However, the variables CWDROWH and CWDCLH are equal to $CWDROW/2+1$ and $CWDCOL/2+1$ respectively. The values of CWDROW and CWDCOL are restricted by subroutines CWCROW and CWCOL such that the product of both values does not exceed $M/3$. This was just an arbitrary choice to keep the correlation window size from becoming excessive. The range of values for the correlation window to be specified is shown in Table I.

After the correlation window size is defined, a temporary "scratch" file is opened for storing the correlation windows surrounding each interesting point using subroutine OPNCWF. This subroutine opens a formatted, sequential scratch file with the default filename. A logical unit number different than the one associated with the image file is automatically assigned to this file to prevent errors.

When the image, processing window, and correlation window sizes have been specified, the intensity array and input buffer initialized, and the correlation window file opened, interesting point processing

can begin. This processing begins when INTPTS calls subroutine IOPSEL. Subroutine IOPSEL requests that the user enter the number corresponding to the interest operator to be used to find the interesting points on the selected image. Three operators are presently available: (1) simple variance (IOP1); (2) directed variance (IOP2); and (3) edged variance (IOP3). IOP4 thru IOP10 are not used. Also, the user may terminate interesting point processing or program execution, if desired. The interest operator number selected is stored in the area COMMON /IOP/ in the variable IOPNBR. This area also contains the variable IPFLAG which flags the storage of the interesting points when set to 1. A detailed discussion of the three interest operators available for processing is given in Chapter III.

Interesting Point Storage. After interesting point processing is complete, if the user has specified interesting point storage, subroutine IFIFLU is called to determine the logical unit number and file name of the file to be used to store the interesting point information. The collection of information stored on the output file is called an interesting point set. As in the case of the image file, subroutine IFIFLU calls subroutines LUNBR1 and FILNM1 to allow the user to interactively enter the logical unit number and file name of the file to be used to store the interesting point set. Similarly, subroutines IFICLK and IFICOR are used for error checking and correction respectively, if needed. The flow diagram of this file description method is identical to the one shown in Figure 8.

All the information necessary to define the interesting point set is assimilated and output to the storage file by subroutine IPSOUT. This subroutine creates the interesting point set file using the logical unit number and file name previously defined in IFIFLU. This file is opened using subroutine OPNIF1 which opens a formatted, sequential file using the assigned file name. Next, the logical records containing the interesting point set information are output to the file. Finally, the interesting point set file is closed using the DISPOSE = 'KEEP' option to save it. The correlation window scratch file previously defined is closed using the 'DELETE' option since it is no longer needed.

Each interesting point set consists of two parts: (1) a header record containing information about the set of interesting points; and (2) the interesting points and their associated correlation windows, stored sequentially in the same order as they are selected. The header record contains five fields: (1) the number of interesting points in the set; (2) the number of the interest operator used to select the interesting points; (3) the correlation window row size; (4) the correlation window column size; and (5) the total correlation window size (product of rows and columns). Each subsequent record contains four items of information. The first three items are the first three record fields which contain the following: (1) the image row location of the interesting point; (2) the image column location of the interesting point; and (3) the value of the interesting point. The last item in the record is the correlation window associated with the

interesting point stored in a single row (i.e., each row of the window placed end-to-end). This window can be reconstructed using the values of the window row and column sizes when the interesting point set is input for matching. The structure of the interesting point set file is shown in Table II.

Since the maximum number of interesting points which may be selected is M (explained later), the maximum number of records in the file is $M + 1$. Also, since the maximum size of each correlation window is restricted to $M/3$, the maximum number of record fields in the file is $M/3 + 3$. The interest operator number, the number of correlation window rows and columns, and the correlation window size are stored in the variables IOPNBR, CWDROW, CWDCOL, and CWDSZE respectively as previously explained. The remaining variables to be output to the interesting point set file are stored within program DIDA in the area COMMON /IPS/. The number of interesting points is stored in the variable IPKNT. The image row and column location, and value of the interesting points, are stored in arrays IPR(M), IPC(M), and VALUIP(M) respectively. Another array, IPCWND(M/3) (also called IPWND), is used as a buffer array for storing and retrieving the values of a single correlation window in the correlation window scratch file.

Therefore, to store the interesting point set, subroutine IPSOUT opens the interesting point set file and writes the header record. The second and subsequent records are then interactively formed by first reading an interesting point correlation window from the correlation

Table II. Interesting Point Set File Structure

RECORD NUMBER	RECORD FIELDS						
	1	2	3	4	5	...	M/3 + 3
1	NUMBER OF INTERESTING POINTS	INTEREST OPERATOR NUMBER	NUMBER OF CORRELATION WINDOW ROWS	NUMBER OF CORRELATION WINDOW COLUMNS	TOTAL CORRELATION WINDOW SIZE (ROWS X COLS)	...	...
2	IP(1) ROW LOCATION	IP(1) COLUMN LOCATION	IP(1) VALUE	IP(1) COR WINDOW (1)	IP(1) COR WINDOW (2)	...	IP(1) COR WINDOW (M/3)
3	IP(2) ROW LOCATION	IP(2) COLUMN LOCATION	IP(2) VALUE	IP(2) COR WINDOW (1)	IP(2) COR WINDOW (2)	...	IP(2) COR WINDOW (M/3)
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
M + 1	IP(M) ROW LOCATION	IP(M) COL LOCATION	IP(M) VALUE	IP(M) COR WINDOW (1)	IP(M) COR WINDOW (2)	...	IP(M) COR WINDOW (M/3)

window file and placing it in the correlation window array buffer IPCWND. The interesting point row location, column location, and value associated with each successive iteration number (up to maximum of M) are then combined with the corresponding correlation window located in the array buffer, and written to the interesting point set file. The number of iterations is determined by the value of IPKNT. After the last record is written, the interesting point set file is closed, and the correlation window scratch file is deleted. A flow diagram illustrating the method of interesting point set storage is shown in Figure 14.

Interesting Point Listing. After the interesting points have been selected and stored (if storage is requested), the user is asked if a listing of the selected interesting points is desired (on the line printer). If an affirmative response is given, the row and column location of the interesting points, and their corresponding values, are listed sequentially in the order selected. If a negative response is given, no listing is obtained.

In summary, to select interesting points the image and processing window sizes are defined first. Next, the image intensity buffer and input buffer are initialized with image data. After this, the user is asked if interesting point storage is desired. If the answer is yes, the correlation window size is defined, the correlation window file is opened, the interesting points are selected using the desired interest operator, and the interesting point set is stored. If storage is not

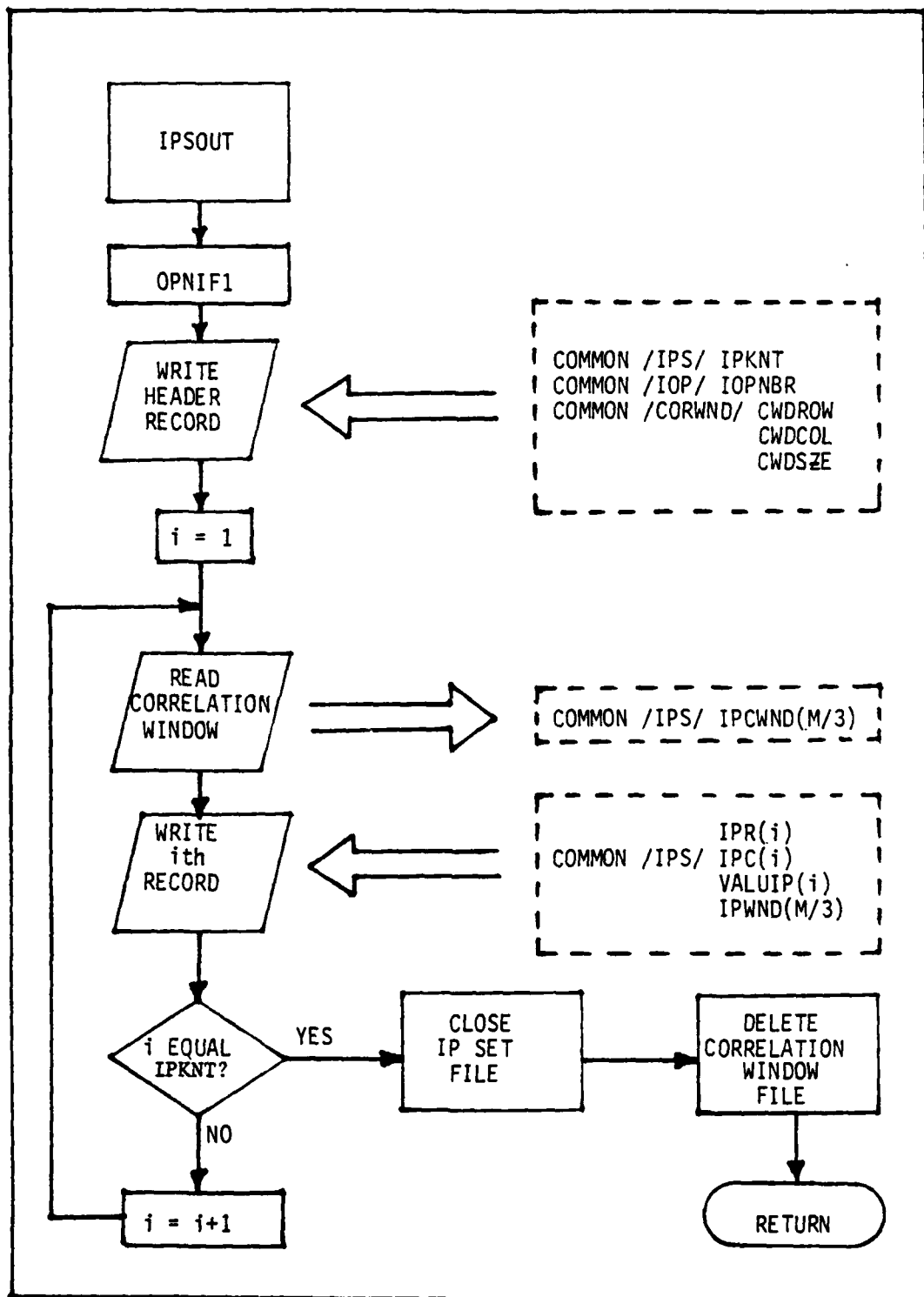


Figure 14. Interesting Point Set Storage Flow Diagram

desired, only the interesting point selection operation is performed. Finally, the user is asked if a listing of the interesting points is desired. If the answer is yes, the points are listed; otherwise, they are not. A flow diagram illustrating the DIDA interesting point selection section is shown in Figure 15.

Interesting Point Matching Section

The subroutine source listings for the interesting point matching section of program DIDA are shown in Appendix E. This section is controlled by subroutine IPMTCH. This subroutine first inputs two interesting point sets for matching. After this, the user selects the matching algorithm desired, and the matching process begins. When matching is complete, the points from each of the two interesting point sets that matched, along with some pertinent match statistics, are listed on the line printer.

To attempt to match the interesting points selected in a certain image to those in another image, two sets of interesting points must be created and stored on a file as discussed in the previous section. It is assumed that these two images are time-dependent, and that objects within the image may be moving with respect to one another and the stationary background. Points in the first image should be matched with those in the second image if and only if they are image plane projections of the same real-world surface point [Ref. 4:334].

To determine if two points match, the matching algorithm must make

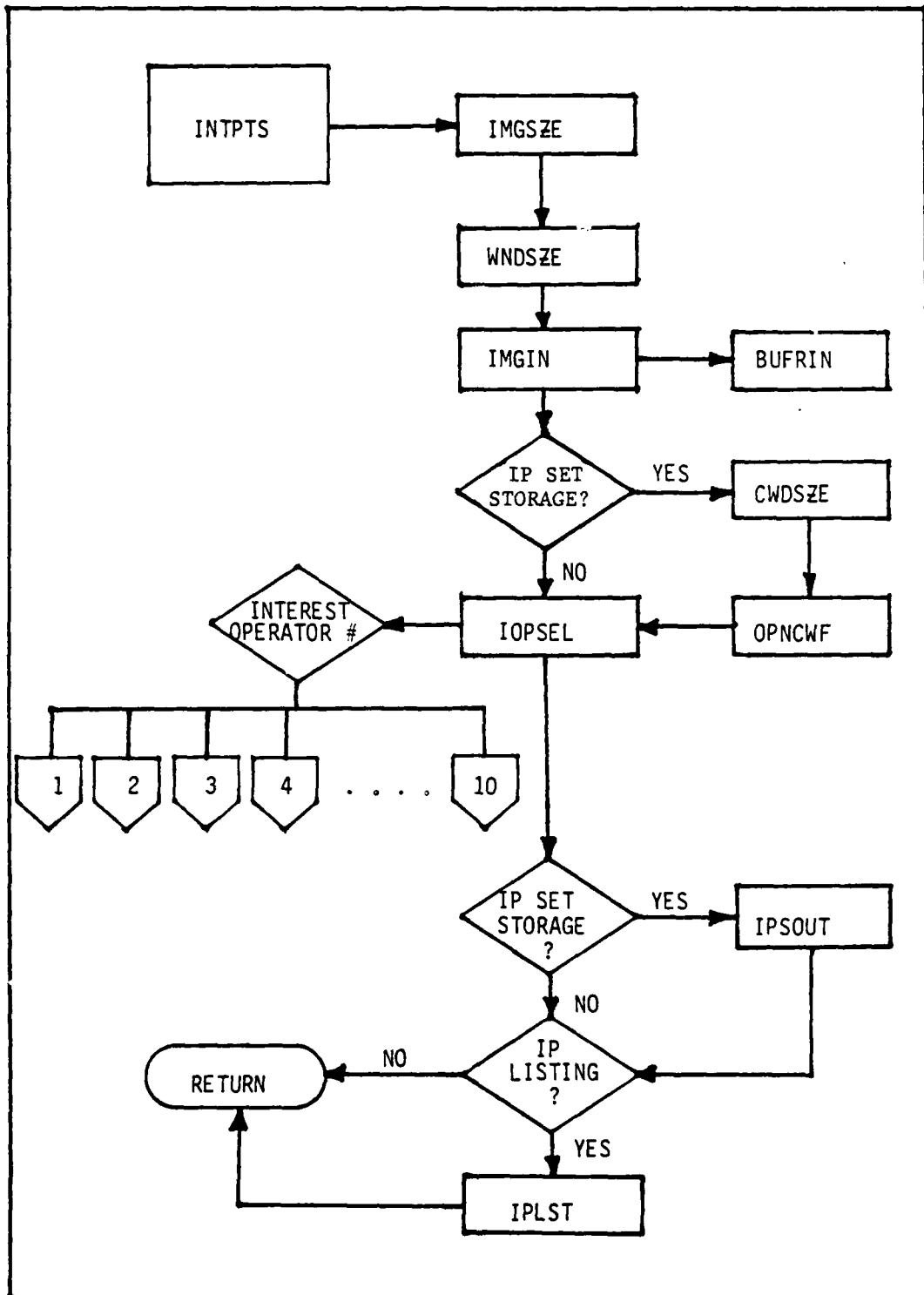


Figure 15. DIDA Interesting Point Selection Section Flow Diagram

some determination as to their similarity. If they are highly similar, then they may be assumed to be matched. However, if several points in close proximity to one another are all similar, it becomes more difficult to determine the two which exactly match. For this reason, all of the interesting points which should be matched may not be matched. Also, false matches may occur. For purposes of this effort, the matching efficiency of the matching algorithm is defined as the ratio of matched to unmatched points expressed as a percentage.

As an illustration, consider the moving object shown in Figure 16. Assume that interesting points were located at each of its corners in image i , and also in image $i + 1$. If the matching algorithm determines that the points in the lower-left hand corner and upper right-hand corner are matched, then two of the four points have been matched. In this case, the matching algorithm has a matching efficiency of 50%.

In the following sub-sections, each of the major operations associated with the interesting point matching section is discussed. As mentioned in the sequence of presentation section of the introduction, the matching algorithm is only generally described in this section. This algorithm is discussed in detail in Chapter IV.

Interesting Point Set Input. Subroutine IPMTCH first calls subroutine IPSOPN to open the files associated with the interesting point sets to be input. To begin with, IPSOPN calls subroutine IF1FLU to allow the user to interactively determine the logical unit number and file name of the first set of interesting points to be input for

matching. This is the same subroutine used to define the logical unit number and file name for storing the interesting point set as described in the interesting point storage sub-section above. Similarly, subroutine OPNIF1 is called to open the file. After the file is open, the header record of the first interesting point set is read and stored in the area COMMON /IPS1/. The information shown in record 1 of Table II is stored in the variables IPKNT1, IPTYP1, CWROW1, CWCOL1, and CWSZE1 respectively. The interesting points and associated correlation windows from interesting point set 1 are now ready to be input.

Subroutine IPSOPN next calls subroutine IF2FLU. The operation of this subroutine is identical to subroutine IF1FLU. Similarly, subroutines LUNBR2, FILNM2, IF2CHK, and IF2COR perform the same functions as before (see Figure 8). The file information for this file is stored in the area COMMON /IFILE2/ in the variables LUN2, NCHAR2, and FNAME2. Subroutine OPNIF2 is then called to open the file, and performs the same function as subroutine OPNIF1. After this second file is open, the header record of the second interesting point set is read and stored in the area COMMON /IPS2/. The information shown in record 1 of Table II is stored in the variables IPKNT2, IPTYP2, CWROW2, CWCOL2, and CWSZE2 respectively. The interesting points and associated correlation windows from interesting point set 2 are now ready to be input.

After the header record of both interesting point sets has been input, a comparison of the information contained in each is displayed

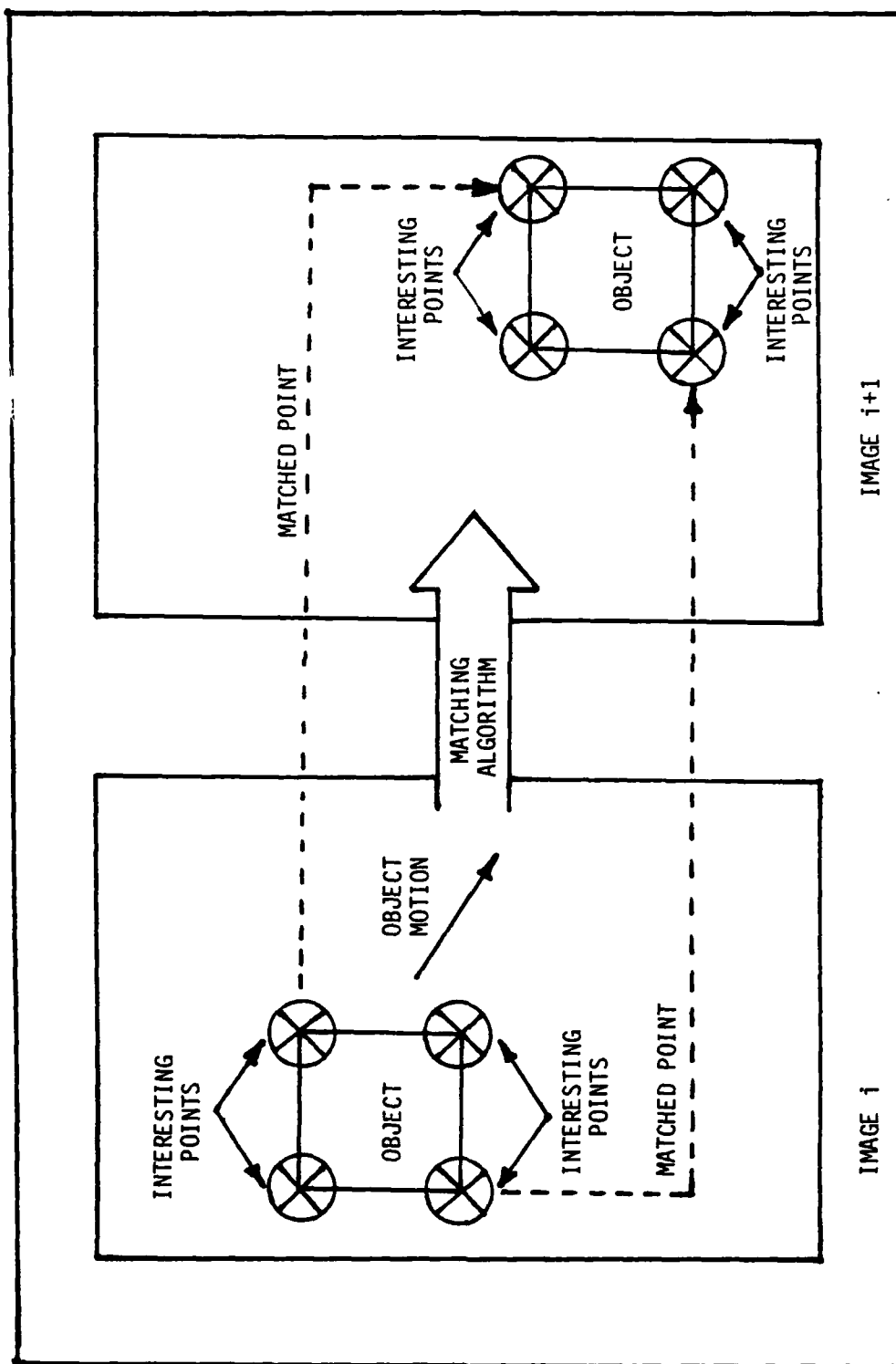


Figure 16. Interesting Point Matching Illustration

to the user. Table III illustrates the information displayed by subroutine IPSOPN. This information is displayed to allow the user to visually check the pertinent information of each interesting point set before matching begins in case this information has been forgotten or lost. The user is next asked if the matching process is to be continued. If an affirmative response is given, matching continues. If a negative response is given, both of the interesting point set files are closed, and program DIDA terminates execution. Notice that it is not necessary for any of the correlation window information to be the same to perform matching. Program DIDA automatically compensates for correlation windows of different sizes by finding the region of common overlap of the two windows and then defining new correlation windows of equivalent sizes containing only the elements common to both original windows (this is described in more detail in Section IV). A flow diagram illustrating the method of interesting point set input is shown in Figure 17.

Interesting Point Matching. If the user desires to continue the matching process after visually comparing the two interesting point sets, subroutine IPMTCH calls subroutine MTASEL. This subroutine allows the user to interactively select the matching algorithm to be used to match the interesting points of the two sets. Presently, only one matching algorithm exists (subroutine MTALG1). Subroutines MTALG2, MTALG3, MTALG4, and MTALG5 are not used. The user may also terminate matching or program execution, if desired.

Table III. Subroutine IPSOPN Information Display

IP SET 1	IP SET 2
NUMBER OF POINTS = IPKNT1	NUMBER OF POINTS = IPKNT2
TYPE INTEREST OP = IPTYP1	TYPE INTEREST OP = IPTYP2
COR WINDOW ROWS = CROW1	COR WINDOW ROWS = CROW2
COR WINDOW COLS = CWCOL1	COR WINDOW COLS = CWCOL2
COR WINDOW SIZE = CWSEE1	COR WINDOW SIZE = CWSEE2

NOTE: IT IS NOT NECESSARY FOR THE CORRECTION WINDOW ROW,
COLUMN, OR OVERALL SIZES TO MATCH TO PERFORM
MATCHING!

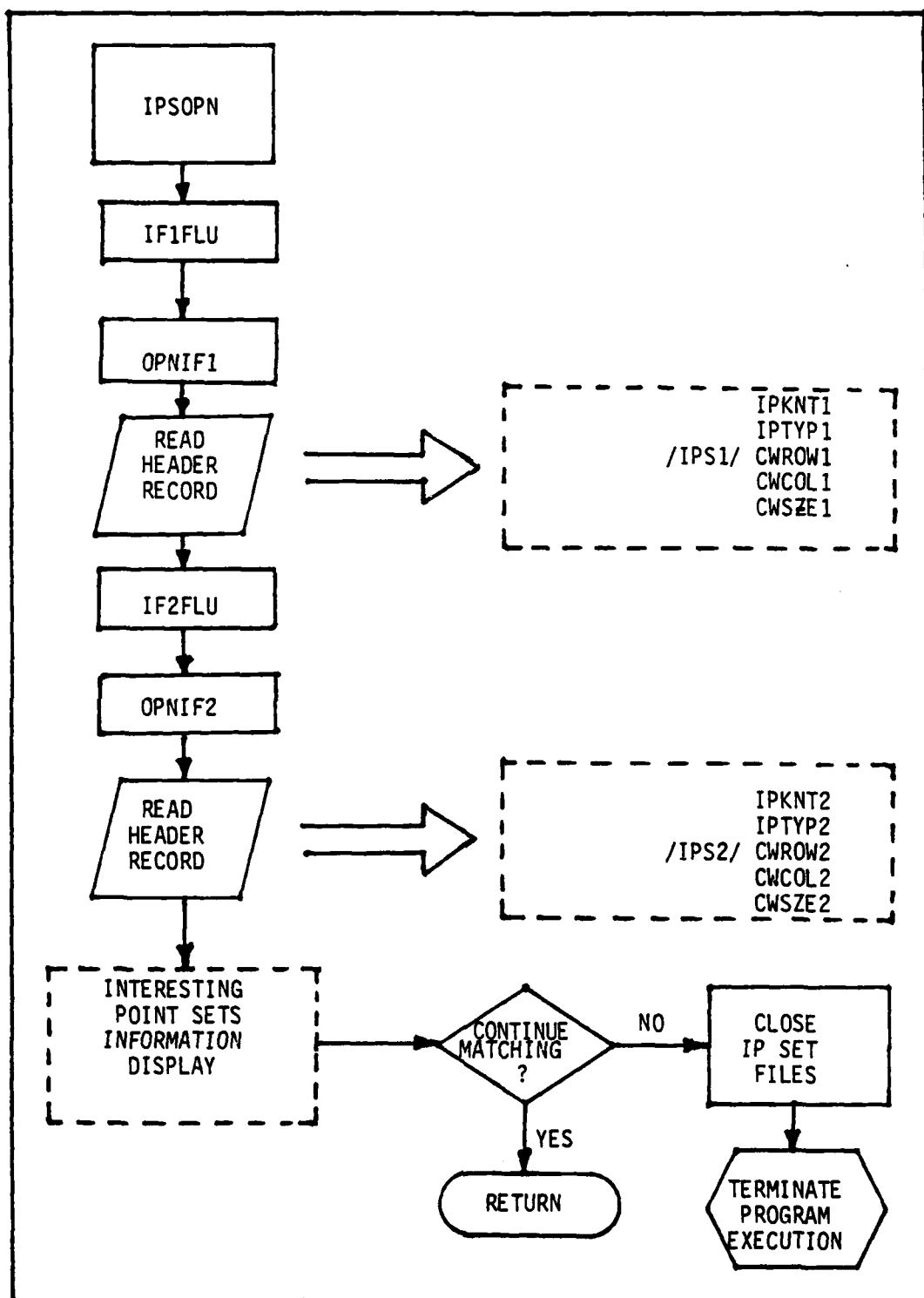


Figure 17. Interesting Point Set Input Flow Diagram

If the user enters a response of 1 when queried by subroutine MTASEL for the desired matching algorithm, subroutine MTALG1 is called. This subroutine begins by describing a disparity window size to be used to form labels from interesting points from each set which are in close proximity of each other. After this, the orientation of the correlation windows surrounding each interesting point in each is determined. A weighting algorithm is then selected to determine the correlation of the two windows for each set of interesting points. From this correlation, a weight is determined. This weight is used to form an initial probability for the label associated with the two interesting points being compared.

Labels are formed for points in each set which are close to each other by taking the difference between the row and column coordinate values of each set of points. In other words, each interesting point in set 1 (now called a node) is compared with all the interesting points in set 2. For all points of set 2 within the disparity window surrounding each node, labels are formed by their coordinate differences. Probabilities are then assigned to these labels using the weights computed by the comparison of their correlation windows. These labels are really the disparity vectors associated with each node.

Once all the labels for all the nodes have been formed, the labels of one node are compared with the labels of other nodes within a small neighborhood surrounding the node to update the initial probabilities assigned to each label. In other words, each node is now compared with

all the other nodes formed by the first interesting point set. For all nodes within a given neighborhood of the node being tested (excluding the test node itself), the labels of each node are compared with the labels of the test node one-by-one. If the disparity vectors of the two labels being compared are similar, the match probability of the label of the test node is increased. If they are dissimilar, the no-match probability is increased.

After several iterations (10 are used for this study), the probability updates are assumed to have reached a steady state. The probabilities of all the labels associated with each node are then checked. If the probability of one of the labels is above 0.7, that node is said to be matched. The point to which it is matched in image 2 is then determined by adding the disparity vector of the label with probability above 0.7 to the coordinates of the node. This determines the coordinates of the point in image 2 to which the node is matched. Thus, the position to which the node moves from image 1 to image 2 has been determined. This, in general, is the essence of disparity analysis. This matching process is described in greater detail in Chapter IV.

Matched Points Listing. After the interesting points from set 1 have been matched to those in set 2, the points which successfully match are listed on the line printer. The node which has been matched and its new location in image 2 are both listed. To accomplish this listing, subroutine IPMTCH calls subroutine MPLST. Subroutine MPLST

also prints out match statistics concerning the results of the matching process. These statistics include: (1) the number of interesting points in image 1; (2) the number of interesting points in image 2; (3) the number of matched points; (4) the number of unmatched points; and (5) the percentage matched (the matching efficiency).

In summary, to match interesting points, the files containing the two sets of interesting points to be matched are opened and compared. If favorable match conditions exist, the matching algorithm is selected and matching begins. Labels are formed from the disparity between points in each set which are close to each other. Probabilities are assigned to these labels based on the comparison of the correlation windows of each interesting point used to form the label. These probabilities are updated by comparing the labels of nodes within a given neighborhood. When these probability updates reach steady state, they are checked to determine if a match has occurred. If so, it is listed on the line printer along with other vital match statistics. A flow diagram illustrating the DIDA interesting point matching section is shown in Figure 18.

Image Display Section

The subroutine source listings for the image display section of program DIDA are shown in Appendix F. This section is controlled by subroutine IMGDSP. As in the image operations and interesting point selection sections, the image size is defined first. Next, the logical unit number and file name of the image to be displayed are

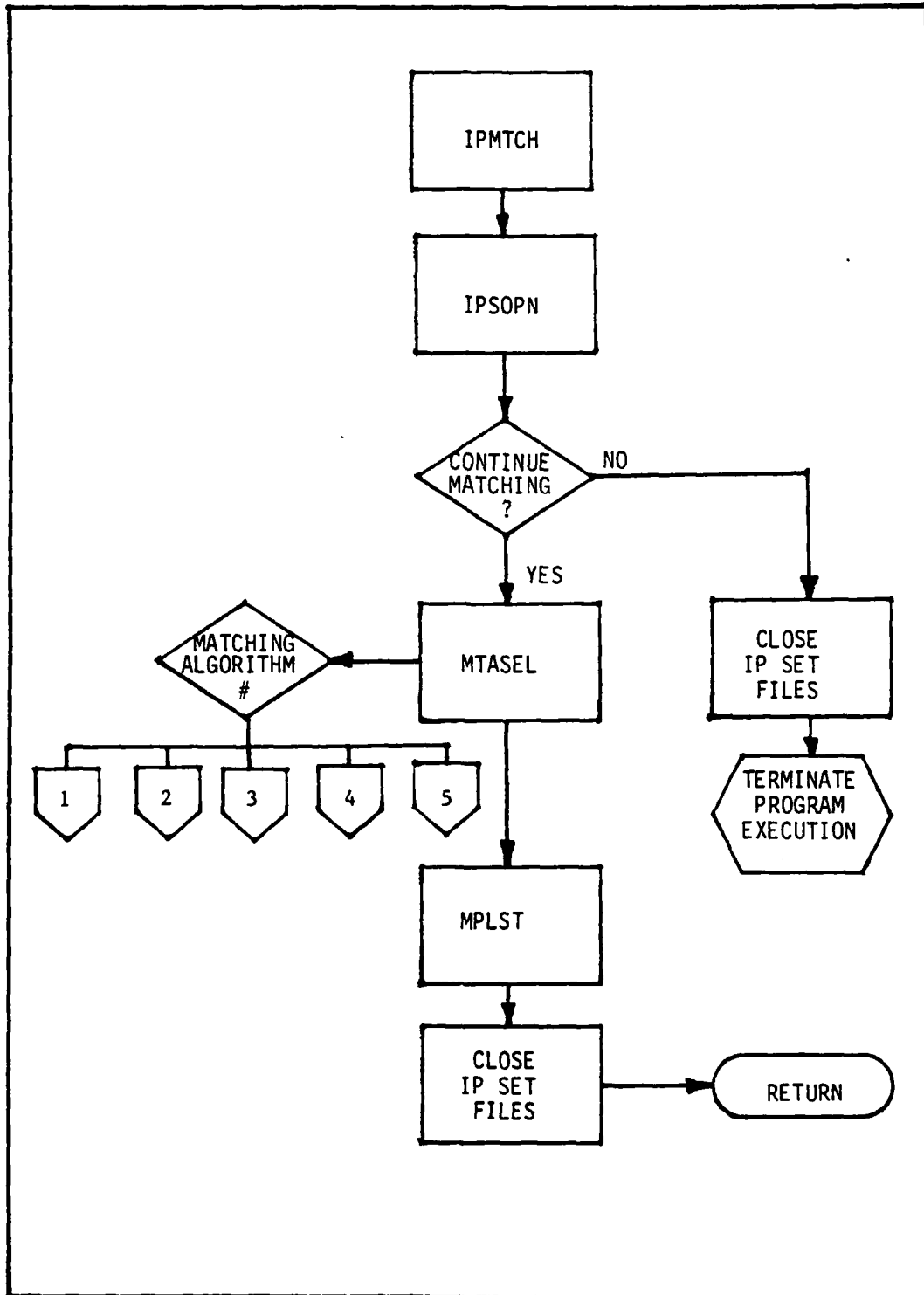


Figure 18. DIDA Interesting Point Matching Section Flow Diagram

interactively defined, and the image file is opened. After this, the user is asked what type of display is desired. When the type of display is selected, this display is sent to a cathode ray tube (CRT) by the RAMTEK driver.

To define the image size, subroutine IMGDSP calls subroutine IMGSE. Subroutines FLUFND and OPNOLD are then called to allow the user to enter the logical unit number and file name of the image to be displayed, and open it. The description of the operation of these subroutines was given in previous sections and will not be repeated here (refer to Figures 6 and 8).

After the image file is opened, and the logical unit number of the RAMTEK driver has been assigned, subroutine IMGDSP calls subroutine DSPSEL to allow the user to select the desired image display operation. Four image display options are available: (1) display a black and white (B/W) image; (2) display a color image; (3) display interesting points on a black and white image; and (4) display interesting points on a color image. The user may also terminate image display operations or program execution, if desired.

B/W Display. To display a black and white image, the user enters the number 1 when queried by subroutine DSPSEL concerning the desired display option. This entry causes the variable ICOLOR to be set to zero, and subroutine RAMST to be called. Subroutine RAMST initially clears the video screen (the CRT), and defines the values of the video look-up table for a black and white image display. The parameters

IPAR, IOSTAT, IVLT, and IMBUF are set to their corresponding initial values; and the initialization parameters are sent to the RAMTEK via the system subroutine call to subroutine WTQIO. A specific description of the operational features of the RAMTEK driver are beyond the scope of this thesis, and will not be discussed further.

After the call to RAMST to initialize the system, subroutine IMDSP1 is called to display the B/W image. This subroutine first asks the user if a blow-up of the desired image from its present size to a size of 512x512 is desired. If an affirmative response is offered, the image is blown-up from its present size to a size of 512x512 using the blow-up feature of the RAMTEK. In this case, the scaling parameter ISCALE is set to zero to signal that blow-up of the image is desired. If image scaling (i.e., blow-up) is not desired, ISCALE is set to 1.

When scaling has been selected, subroutine IMDSP1 inputs the image to be displayed one row at a time by calling subroutine INPUT. The input byte intensity values are then transferred to the integer word intensity buffer by performing byte-to-word number conversion (see Appendix G). These rows of intensity data are then sent to the RAMTEK driver by calling subroutine RAMWT. After the image has been displayed, the image file is closed by calling subroutine CLOSKP.

Subroutine RAMWT first checks the value of ISCALE to see if scaling is desired. If it is, the image pixels displayed by the RAMTEK are expanded by the appropriate scale factor. Next, the RAMTEK instruction set is established based on the scaling parameters

specified. The image data is then sent to the RAMTEK driver via subroutine WTQIO, and a B/W image is displayed.

Color Display. To display a color image, the user enters the number 2 when queried by subroutine DSPSEL concerning the desired display option. This entry causes the variable ICOLOR to be set to 1, and subroutine RAMST to be called. Again, subroutine RAMST initially clears the video screen, but this time the video look-up table is defined for a color rather than a B/W image. From this point on, the image display sequence is identical to that for the B/W display. The change in the sub-channels of the video look-up table produces the color display.

B/W Interesting Points Display. To display a B/W image with a set of interesting points superimposed on it, the user enters the number 3 when queried by subroutine DSPSEL concerning the desired display option. This entry causes the variable ICOLOR to be set to zero, and subroutine RAMST to be called, as explained above. Next, subroutine IMDSP2 is called to display the interesting points on the B/W image.

Subroutine IMDSP2 first calls subroutine IMDSP1 to display the desired B/W image, as explained above. After this, the user is asked to enter the logical unit number and file name of the set of interesting points to be written over the displayed image. Subroutine IFIFLU is then called to allow the user to interactively input this information, and the interesting point set file is opened by calling subroutine OPNIF1. Next, the number of interesting points in the set

is read, and the interesting points are read in from the file and stored in arrays IPR(M) and IPC(M) in the area COMMON /IPS/. Subroutine RAMIP is then called to write the interesting points on the displayed image, and the interesting point set file is closed.

Subroutine RAMIP sets up the RAMTEK instruction set by modifying the row and column location of the already displayed image at each location where an interesting point is to be displayed. A 3x3 square of a contrasting intensity is written about the point on the CRT where the interesting point exists. To accomplish this, yellow squares are written on a B/W image (the case here) and black squares are written on a color image. Again, after the instruction set has been set up, subroutine WTQIO is called to send the data to the RAMTEK driver.

Color Interesting Points Display. To display a color image with a set of interesting points superimposed on it, the user enters the number 4 when queried by subroutine DSPSEL concerning the desired display option. This entry causes the variable ICOLOR to be set to 1, and subroutine RAMST to be called, as explained above. Next, subroutine IMDSP2 is called to display the interesting points on the color image. From this point on, the image display sequence is identical to that for the B/W interesting points display. A flow diagram illustrating the image display section of program DIDA is shown in Figure 19.

Image Data Bases

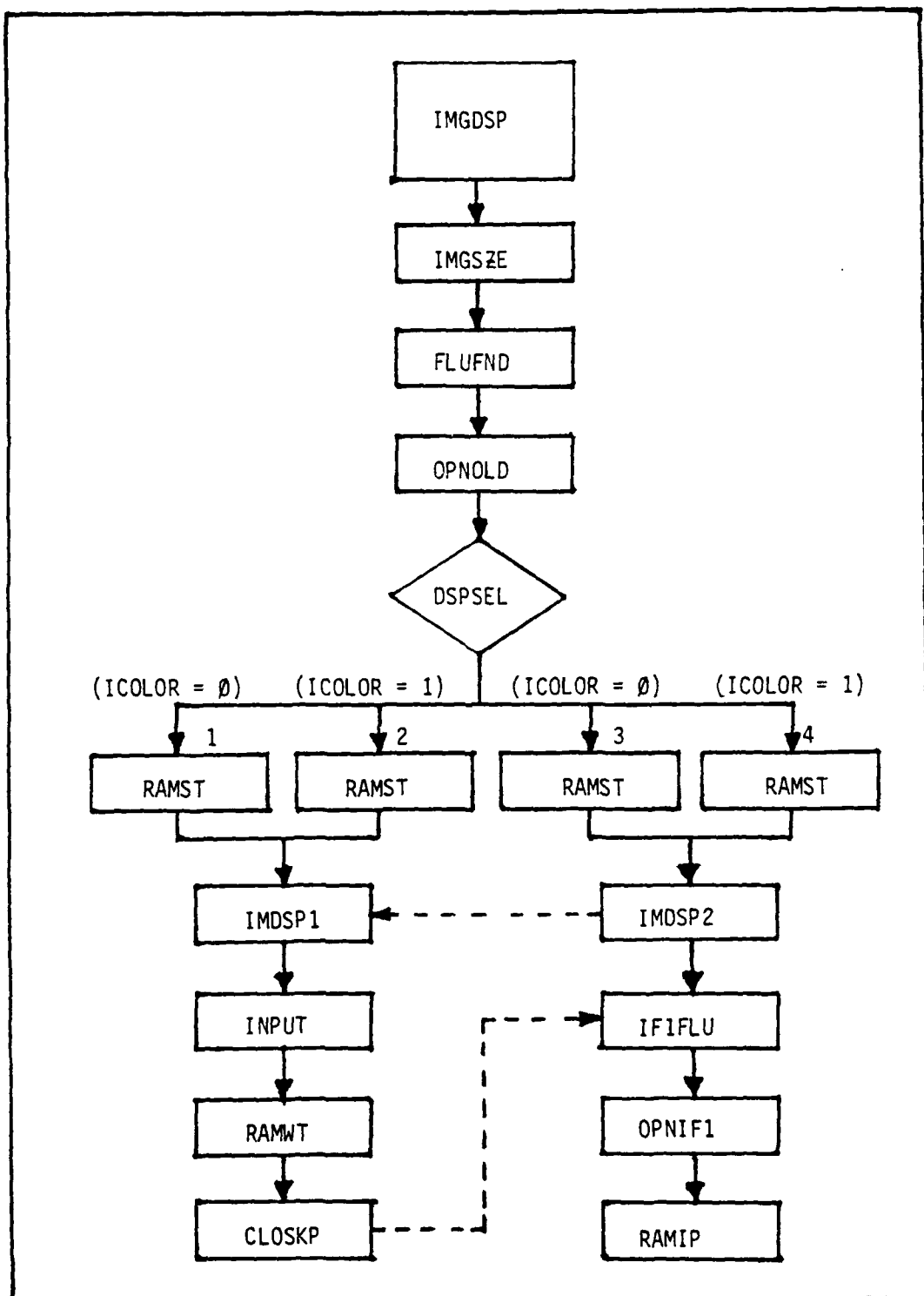


Figure 19. DIDA Image Display Section Flow Diagram

To accomplish the image operations, interesting point selection, interesting point matching, and image display as discussed in the previous sections, one common ingredient is needed - an image! To obtain images for processing, three data bases were created on disk files from data tapes secured from three independent sources. These data tapes contain digitized image data from real-world (or simulated real-world) scenes as discussed in the image size subsection of the image operations section above (also see Figure 5). A description of each of these data bases, along with a brief discussion concerning the method used to transfer them from tape to disk, is given in the following subsections.

NVL Data Base. The first data base to be discussed is a series of 20 noise-corrupted images received from the Lockheed Palo Alto Research Laboratory [Ref. 9]. This imagery was created as part of the DARPA Image Understanding Program under the direction of Lt. Col. L. E. Druffel, USAF, and monitored by Dr. Tamburino. The imagery was digitized from a simulated flight over the terrain board of the U.S. Army Night Vision Laboratory (NVL), heading due north over villages east of the river. Data defects include banding, graininess, and poor focus.

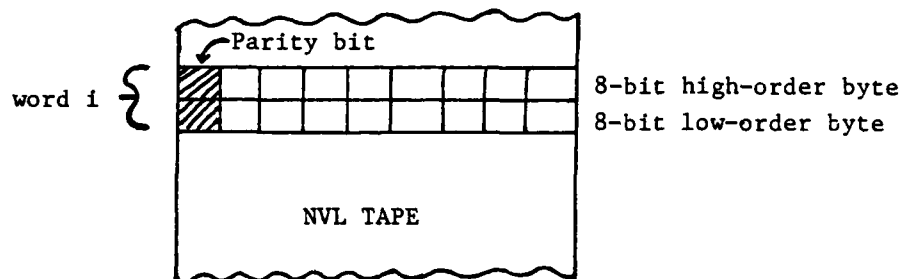
The images were received on a 9-track magnetic tape, 1600 bits/inch (bpi), phase encoded, odd parity. Size of each image is 512x512 pixels. Each image file on the tape contains 129 physical records. The first record is the header, and records 2 - 129 contain

the image data.

Each physical record contains 2052 16-bit words as described below:

	WORDS	IMAGE ROW
	1 - 512	$4n - 7$
	513 - 1024	$4n - 6$
Description of record n	1025 - 1536	$4n - 5$
$(2 \leq n \leq 129)$	1537 - 2048	$4n - 4$
	2049 - 2052	UNUSED

Thus, each physical record contains four image rows. For example, for record 20 ($n = 20$), the image rows contained are $4(20) - 7 = 73$, $4(20) - 6 = 74$, $4(20) - 5 = 75$, and $4(20) - 4 = 76$. Each intensity within each record is stored as a 16-bit word represented by two 8-bit bytes to produce a range of intensities from 0 to 511 as shown below:



As mentioned previously, an 8-bit byte is required to represent the range of numbers from 0 to 255. To represent the range from 0-511, only one more bit is required. Thus, the high order byte for each word is only 1 or 0. It will be a 1 if the intensity value is greater than 255 (also see Appendix G).

In order to transfer this data base to a disk file and make it compatible with the input requirements of program DIDA, a data conversion program was written. This program is a modification of the standard tape-to-disk conversion program, and is shown in Appendix H. To convert the data, it was necessary to write it to the disk files as a direct access, unformatted, sequential file in the range of 0 to 255. Thus, data scaling was required.

To scale the data, the low-order byte is tested for a negative number. If negative, it is divided by two (to scale it) and the sign bit is changed (see Appendix G). If non-negative, the number is merely divided by two. Next, the high-order byte is tested. If it is equal to 1, this 1 is shifted into the MSB position of the low-order byte to preserve the range integrity. If the high-order byte is zero, no other operation is performed.

After this number conversion and scaling is accomplished, the high-order bytes are stripped away from the low-order bytes (which now contain the correct converted scaled intensity values in the range 0-255) and output to the disk file. Nine of the 20 image files were written to the disk before its storage capacity was exceeded. A listing of the disk files, and other pertinent information, is shown in Table IV.

UM Data Base. The second data base to be discussed is a sequence of 18 frames (images) of various objects received from Dr. William B. Thompson of the University of Minnesota, Computer Science

Department, Minneapolis, MN. The tape received contains a collection of image pairs and sequences from their local library. Five groups of images were received.

The images received were recorded on a 9-track tape at a density of 1600 bpi. Each image is contained in a separate file. One tape record exists for each image row, and each record field corresponds to a single 8-bit gray scale intensity value in the range from 0 to 255. Thus, unlike the NVL data, this data base was received in a format already compatible with the input format of program DIDA.

All that was necessary to transfer this data base to disk was to use the standard tape-to-disk transfer program to read the data from the tape and transfer it to a direct access, unformatted, sequential file on the disk. This transfer program is similar to the one shown in Appendix H, so it is not shown to conserve space.

As mentioned above, the images on the tape were contained in five groups:

- (1) A truck which moves against a stationary background.
- (2) A simulated fly-over of downtown Minneapolis.
- (3) A stereo pair of pictures which include a person sitting in a chair.
- (4) An 8-frame sequence of childrens toys. Two of the toys move in opposite directions. In the later frames, one of the toys partially

occludes the other.

(5) A 4-frame sequence of a room taken with a side-looking moving camera.

All images were 128x128 pixels. A listing of the disk files, and other pertinent information, is shown in Table IV.

UH Data Base. The final data base to be discussed is a 33 frame sequence of images received on tape from Dr. Wesley E. Snyder of the Computer Science Department of North Carolina State University, Raleigh, NC. This tape was obtained by Dr. Snyder from professor Hans Nagel of the University of Hamburg, Hamburg, Germany. Each tape image contains 512 records of one image row per record. Each record contains 512 8-bit data intensity values in the range from 0 to 255. Thus, like the UM data base, this data base was received in a format already compatible with the input format of program DIDA.

Transfer of this data base from tape-to-disk was quite simple. The tape was created on a DEC VAX machine using the "copy" command. As it turns out, the copy command of the DEC PDP 11/45 is compatible with that of the VAX. Thus, all that was required to copy to the disk was to use the PIP "COPY" command and the data was transferred.

The UH data base consists of a 33 frame sequence of a traffic scene. In the sequence, a taxi cab is shown turning a corner at a busy intersection in Hamburg. Only eight frames were copied to disk before the disk storage space limit was exceeded. However, this was more than

sufficient for purposes of this study. A listing of the disk files, and other pertinent information, is shown in Table IV.

In summary, three data bases were created. The NVL data base contains a noise-corrupted set of images taken from a terrain board during a simulated flight. The UM data contains various pairs of images and image sequences taken from a moving truck, a simulated flyover of downtown Minneapolis, a person sitting in a chair, children's toys moving relative to each other, and a room taken with a moving side-looking camera. Finally, the UH data base contains a traffic scene showing a taxi cab turning left at a busy intersection in Hamburg.

Table IV. Image Data Base Characteristics

NVL DATA BASE Disk 5 UIC = [10,20]		UM DATA BASE Disk 4 UIC = [10,15]		UH DATA BASE Disk 3 UIC = [10,10]	
FILE NAME	SIZE	FILE NAME	SIZE	FILE NAME	SIZE
TERAIN.IMG;1	512 X 512	TRUCK.IMG;1	128 X 128	TRAFIC.IMG;1	512 X 512
TERAIN.IMG;2	512 X 512	TRUCK.IMG;2	128 X 128	TRAFIC.IMG;2	512 X 512
TERAIN.IMG;3	512 X 512	FLYOVR.IMG;1	128 X 128	TRAFIC.IMG;3	512 X 512
TERAIN.IMG;4	512 X 512	FLYOVR.IMG;2	128 X 128	TRAFIC.IMG;5	512 X 512
TERAIN.IMG;5	512 X 512	PERSON.IMG;1	128 X 128	TRAFIC.IMG;6	512 X 512
TERAIN.IMG;6	512 X 512	PERSON.IMG;2	128 X 128	TRAFIC.IMG;7	512 X 512
TERAIN.IMG;7	512 X 512	TOYS.IMG;1	128 X 128	TRAFIC.IMG;10	512 X 512
TERAIN.IMG;10	512 X 512	TOYS.IMG;2	128 X 128	TRAFIC.IMG;11	512 X 512
TERAIN.IMG;11	512 X 512	TOYS.IMG;3	128 X 128	NOTE: TRAFIC.IMG;4 not used due to a parity error on the tape.	
		TOYS.IMG;4	128 X 128		
		TOYS.IMG;5	128 X 128		
		TOYS.IMG;6	128 X 128		
		TOYS.IMG;7	128 X 128		
		TOYS.IMG;10	128 X 128		
		ROOM.IMG;1	128 X 128		
		ROOM.IMG;2	128 X 128		
		ROOM.IMG;3	128 X 128		
		ROOM.IMG;4	128 X 128		

III. Interesting Point Selection Algorithms

Computer analysis of time-varying images, or CATVI, implies the processing of a time-sequence of images in order to extract some property which changes in time [Ref. 10:7]. As an example of this, consider the moving tank mentioned on page 1 of the introduction section. As the tank moves from scene-to-scene through the imaging sensors field-of-view, it creates a displacement relative to the background (and sensor) due to its motion. Thus, as the sensor processes sequential images, the motion of the tank creates a difference in the information content of the images from one frame to the next. When such differences occur between two images, a "disparity" is said to exist between them, which is represented as a vector field mapping one image into the other [Ref. 4:333]. The determination of disparity is often referred to as the "correspondence problem" [Ref. 11:399]; or, more simply, "disparity analysis."

The goal of disparity analysis is to assign disparities (which are two-dimensional vectors for the purpose of this study) to a collection of points in one of the images. These points, called "interesting points," are located (selected) using an interesting point selection algorithm called an "interest operator." Once these interesting points have been selected on a given image, they are matched to their corresponding points on a subsequent image using a matching algorithm. As mentioned in the interesting point matching section of Chapter II, not all of the interesting points selected will necessarily be matched.

Also, ambiguous matches may occur (i.e., interesting points in one image matched to the wrong point in another image), or points may be occluded in some images and visible in others. To be effective, an interest operator must select interesting points which may be easily matched from image-to-image. Thus, interest operators must be sensitive to local variances, edges, and other distinctive image properties in order to select potentially matchable points. It is also advantageous to select points which are easily distinguishable from their neighbors.

Selection of interest operators varies widely. Hannah suggests interest operators with variances such that the interesting points selected are defined to be local maxima within the region of interest [Ref. 12:202-203]. Barnard and Thompson describe a sum-of-squares differencing in four directions over the processing window area [Ref. 4:335]. This is the technique first defined by Moravec [Ref. 13: 584], and has become known as the "Moravec Operator." Gennery also uses this Moravec Operator in a stereo vision system for an autonomous vehicle [Ref. 14: 576]. Various other interest operators are described more generally in a survey article by Martin and Aggarwal [Ref. 15].

In the sections to follow, the three interest operators used in this study are discussed in detail. These three operators are: (1) simple variance; (2) directed variance; and (3) edged variance. The second operator is the Moravec Operator. These three operators are summarized by Hannah [Ref. 9: (3-2)-(3-5)]. A description of the

implementation in program DIDA is given along with the discussion of each interest operator.

Simple Variance Interest Operator

As mentioned above, interesting points are selected based on some "interesting" feature of various portions of the image. As the processing window of size WNDROW X WNDCOL shown in Figure 10 is moved across the image as shown in Figure 12, the area of the image enclosed by the window is searched for an interesting feature which may be used to define an interesting point. One method of defining an interesting point is to use the statistical variance (referred to as "simple" variance) over the window as an interest operator. If the simple variance of a given window position exceeds a specific user-defined threshold, the center of that window position is defined to be an interesting point.

Consider an $n \times m$ dimensional array of numbers called A . Each element of array A is located by its row and column location i and j , respectively, and is referred to as a_{ij} . The simple variance is the average of the square of the deviations of A about their mean [Ref. 16: 7]. The mean, $\bar{M}$, is defined as

$$\bar{M} = \frac{1}{nm} \left\{ \sum_{i=1}^n \sum_{j=1}^m a_{ij} \right\} \quad (1)$$

The variance, V , is given by

$$V = \frac{1}{nm} \left\{ \sum_{i=1}^n \sum_{j=1}^m (a_{ij} - \bar{M})^2 \right\} \quad (2)$$

Or, more simply as [Ref. 16: 13]

$$V = \frac{1}{nm} \left\{ \sum_{i=1}^n \sum_{j=1}^m (a_{ij})^2 - \frac{1}{nm} \left[\sum_{i=1}^n \sum_{j=1}^m a_{ij} \right]^2 \right\} \quad (3)$$

$$= \frac{1}{nm} \left\{ \sum_{i=1}^n \sum_{j=1}^m (a_{ij})^2 \right\} - \left\{ \frac{1}{nm} \left[\sum_{i=1}^n \sum_{j=1}^m a_{ij} \right]^2 \right\} \quad (4)$$

$$= \frac{1}{nm} \left\{ \sum_{i=1}^n \sum_{j=1}^m (a_{ij})^2 \right\} - \bar{M}^2 \quad (5)$$

Thus, to compute the variance, the mean is first computed by summing up all the elements of A, then dividing by the total number of elements. The variance is then calculated by summing up all the squares of the elements of A, dividing this sum by the total number of elements, and subtracting the square of the mean from the result.

In program DIDA, to select interesting points using the simple variance operator, the user enters a 1 when queried concerning the desired interest operator in subroutine IOPSEL, as explained in the interesting point selection section of Chapter II above. This entry causes IOPSEL to call subroutine IOPI which computes the variance of each window position, and then determines if an interesting point exists. If an interesting point is found to exist at a certain window location, it is placed in arrays IPR, IPC, and VALUIP in the area COMMON /IPS/. If interesting point set storage has been requested, the correlation window corresponding to the interesting point is stored on

a temporary scratch file, as previously explained.

To begin with, subroutine IOP1 calls subroutine THRHL to allow the user to specify the variance threshold to be used to test for interesting points. Subroutine THRHL also allows the user to check the threshold entry (specified by the variable THRESH) and to make corrections, if desired. Next, subroutine INITAL is called to initialize various parameters associated with interesting point processing. To select the interesting points, subroutine SVAR is called which returns the value of simple variance for a given window position. This value of variance (specified by the variable VARNCE) is then tested against the user-defined threshold. If the value of VARNCE is greater than the value of THRESH, an interesting point exists at that window location, and subroutine IPSTOR is called to store the location and value of the interesting point (and the correlation window, if requested). Finally, subroutine WNDMOV is called to advance the window to the next sequential row and column position (see Figure 12). If the last window position is detected by subroutine WNDMOV (Fig. 12(1)), interesting point processing is complete and subroutine CLOSKP is called to close the image file.

Simple Variance Computation. To compute the variance at each window position, subroutine SVAR calls subroutine MEANFD to compute the mean as given by equation (1). The sum of the squares of the intensity elements for each window position is computed in subroutine MEANFD,

along with the sums of the intensities. The sum of the intensities, the sum of the squares, and the value of the mean are stored in the area COMMON /MEAN/ by the variables COLSUM, SQRSUM, and MEAN respectively. SVAR computes the variance by squaring the value of MEAN obtained in MEANFD and subtracting the result from the product of SQRSUM and WNDINV as given by equation (5).

To compute the mean at each window position, subroutine MEANFD sums the values of the intensity elements of the window and multiplies the resultant sum by WNDINV. To determine these window sums, partial sums of each window column are first formed. These partial sums are then summed to form the total window sum. From equation (1), let

$$S_j = \sum_{i=1}^n a_{ij} \quad (j=1,2,3,\dots,m) \quad (6)$$

then equation (1) can be rewritten as

$$\bar{M} = \frac{1}{nm} \left\{ \sum_{j=1}^m S_j \right\} \quad (7)$$

Upon closer inspection, an even further simplification is observed. Recall that the window is moved across the image one column at a time horizontally. Thus, after summing all the column sums of the window when it is in its leftmost position on the image, subsequent sums can be formed by adding the column sum to the immediate right of the window to the previous sum, and subtracting the leftmost column sum of the window. Thus, if $\bar{M}_1$ is the mean of the window in its leftmost

AD-A127 409

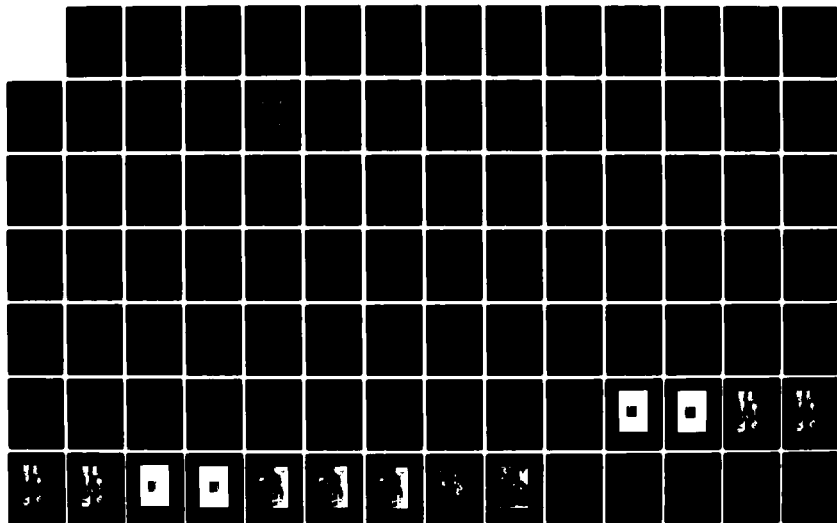
DISPARITY ANALYSIS OF TIME-VARYING IMAGERY(U) AIR FORCE
INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF
ENGINEERING F D COOPER DEC 81 AFIT/GEO/EE/81D-1

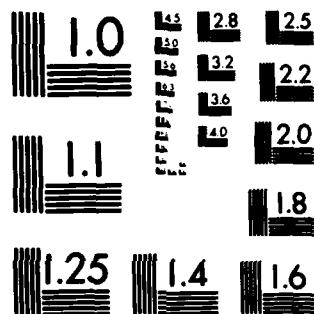
2/4

UNCLASSIFIED

F/G 20/6

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

position on the image, the mean of the window displaced one column to the right, $\bar{M}_2$, is given by (from equations (6) and (7))

$$\bar{M}_2 = \bar{M}_1 + S_{m+1} - S_1 \quad (8)$$

Similarly, the sum of the squares of the intensities over the window can be represented by partial sums as

$$T_j = \sum_{i=1}^n (a_{ij})^2 \quad (j = 1, 2, 3, \dots, m) \quad (9)$$

If $\bar{N}_1$ is the sum of the partial square sums of the window in its leftmost position on the image, the sum of the squares of the window displaced one column to the right, $\bar{N}_2$, is given by

$$\bar{N}_2 = \bar{N}_1 + T_{m+1} - T_1 \quad (10)$$

Now consider the window in its upper left corner position on the image as shown in Figure 12(a). Before processing begins, subroutine MEANFD calls subroutine ADDCOL. This subroutine forms partial sums of the intensities and intensity squares of each column of the image. Each of these columns is of length WNDROW. Thus, IMGCOL partial sums of length WNDROW will be formed by subroutine ADDCOL before the window is moved from its leftmost position on the image.

When processing begins, the first WNDROW partial sums are summed to obtain the element sums and sum-of-squares of all the elements within the window (see Figure 10), and these sums are stored in the

variables COLSUM and SQRSUM respectively. As the window moves from column to column across the image, COLSUM and SQRSUM are recomputed according to equations (8) and (10) respectively until the last column is reached. This process is repeated each time the window is relocated at its leftmost position on the image.

Window Movement. In order to move the window across the image, subroutine WNDMOV is called. Each time this subroutine is called, the window is advanced horizontally one column until the rightmost column of the window is aligned with the rightmost column of the image. On the next call to WNDMOV, the window is moved back to the position where the leftmost column of the window is aligned with the leftmost column of the image, and down one row. This process continues until the rightmost column of the window is aligned with the rightmost column of the image (IMGCOL), and the bottom row of the window is aligned with the last row of the image (IMGROW). At this point, the flag IMGEND is set to 1 to signal the calling program to terminate processing. This window movement is shown pictorially in Figure 12.

In order to move the window across the image, four variables are required to specify the location of the window with respect to the image. These variables are stored in the area COMMON /MOVE/ and are designated ROWKNT, COLKNT, ROWDIF, and COLDIF. ROWKNT and COLKNT specify the top row and left column of the window respectively. ROWDIF and COLDIF specify the bottom row and right column of the window respectively. Initially, ROWKNT and COLKNT are set to 1, and ROWDIF

and COLDIF are set to the values of WNDROW and WNDCOL respectively. As the window is moved horizontally across the image, COLDIF is incremented by COLKNT + WNDCOL and COLKNT is incremented by 1 for each new column position. When COLDIF is equal to IMGCOL when WNDMOV is called, ROWDIF is incremented by ROWKNT + WNDROW, ROWKNT is incremented by 1, and COLDIF is set equal to WNDCOL and COLKNT is set equal to 1 again. Finally, when ROWDIF is equal to IMGROW and COLDIF is equal to IMGCOL, the window is at its final position on the image (Figure 12(1)) and IMGEND is set to 1 to signal the calling program that the end of the image has been attained. A flow diagram illustrating the method of window movement is shown in Figure 20.

When subroutine ADDCOL is called, the partial sums for each column of length WNDROW are formed across the entire image. Thus, there will be IMGCOL partial sums. These sums are stored in the area COMMON /REG1/ in the arrays S(M) for the element partial sums, and T(M) for the partial sums of the element squares. In subroutine MEANFD, each time the window is in its leftmost position on the image, subroutine ADDCOL is called to form the partial sums and the first WNDCOL sums are added across the window to form the initial values of COLSUM and SQRSUM. The mean is computed by the product of COLSUM and WNDINV, and SQRSUM is used by subroutine SVAR to compute the variance.

As the window is advanced horizontally across the image, the values of COLSUM and SQRSUM are recursively updated. These updates are formed by adding the partial sums just to the right of the previous

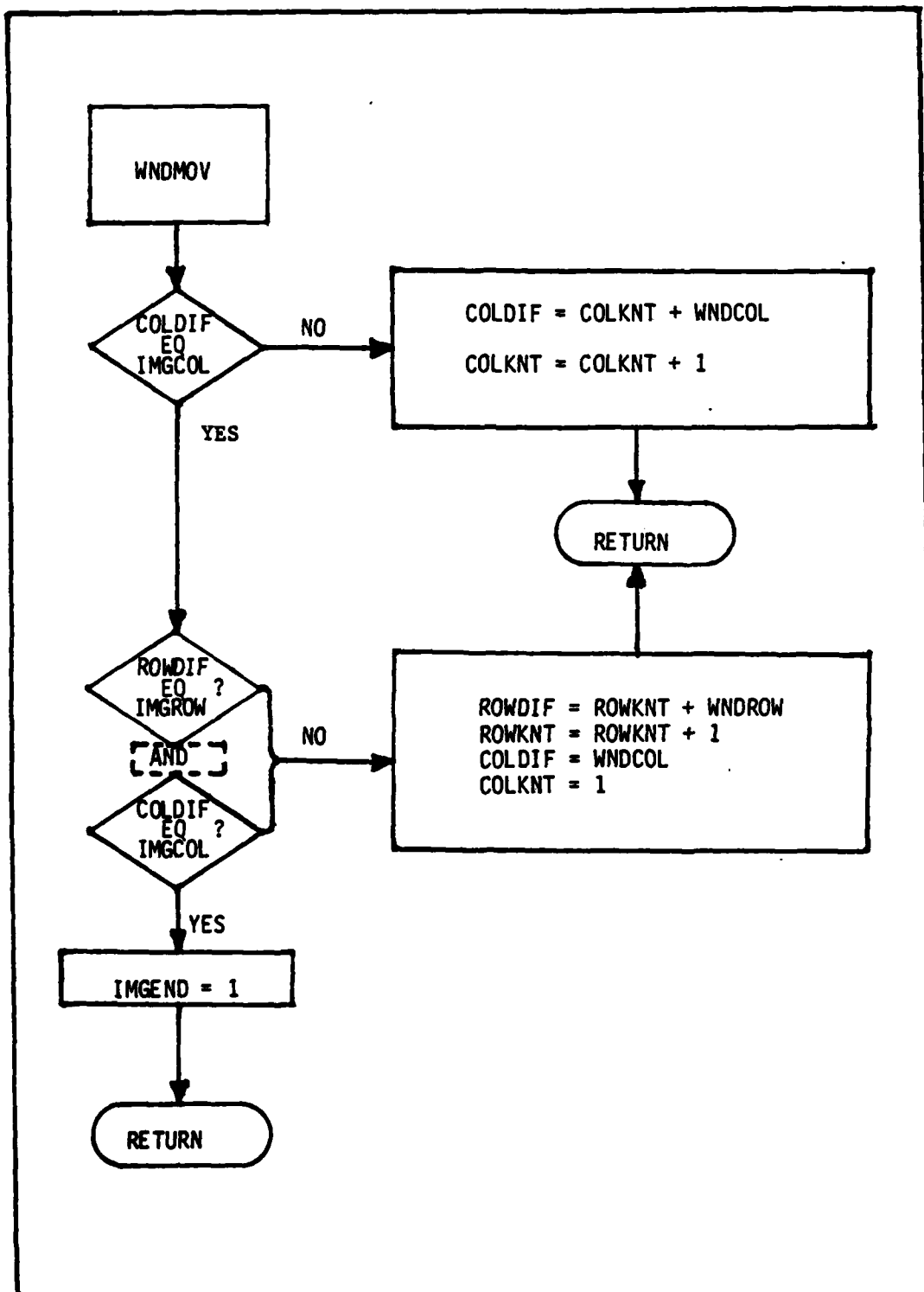


Figure 20. Window Movement Flow Diagram

window position (given by the present value of COLDIF) to the values of COLSUM and SQRSUM, and subtracting the leftmost column sums of the previous window position (given by the value of COLKNT-1). Thus, the update equations are given by

$$\begin{aligned} \text{COLSUM} &= \text{COLSUM} + \text{S}(\text{COLDIF}) - \text{S}(\text{COLKNT}-1) & (11) \\ \text{SQRSUM} &= \text{SQRSUM} + \text{T}(\text{COLDIF}) - \text{T}(\text{COLKNT}-1) & (12) \end{aligned}$$

Row Change Operation. After the window has moved across the image the first time and advances to the second row, the first row of intensity data in the intensity array INT is replaced with the first row of data from the input buffer IOBUFR as shown in Figure 13(b). In subroutine MEANPD, after the first pass of the window across the image (ROWKNT > 1), subroutine ROWCHG is called each time the window returns to its leftmost position, and processing halts until the row change operation has transpired.

Subroutine ROWCHG replaces a row of data in the intensity array INT with the appropriate row of data from the input buffer IOBUFR. After this row change has taken place, byte-to-word number conversion is accomplished to insure that the correct intensity values are transferred (see Appendix G). As may be seen by examining the sequence of row change operations shown in Figure 13, the row number of the intensity array will not always directly correspond to the row number of the input buffer. To keep track of this difference, two row pointers are used. For the intensity array, the pointer WNDKNT points to the new location of where the next row of image data should be

placed. This pointer is reset to 1 each time the value of WNDKNT is equal to the value of WNDROW (it is actually set to zero, but becomes 1 when incremented). This resetting of WNDKNT is shown in Figure 13 (i) and (j).

For the input buffer array, the pointer BUFKNT points to the new location of where the next row of image data should come from when transferred to the intensity array. This pointer is reset to 1 each time the value of BUFKNT is equal to the value of L (assuming the end of the image has not been reached). This resetting of BUFKNT is shown in Figure 13 (e), (f), and (g). The pointers WNDKNT and BUFKNT, along with the value of the current record being processed (specified by the value of RECNUM), are stored in the area COMMON /COUNT/.

After the last row of data is transferred from the input buffer to the intensity array, this buffer is refilled with L more rows of data from the image file as shown in Figure 13 (f). This is accomplished when subroutine ROWCHG calls subroutine BUFRIN. Subroutine BUFRIN adds L to the value of the last record input and inputs that many new rows of image data into the input buffer IOBUFR. This is accomplished by calling subroutine INPUT. Subroutine BUFRIN also tests the value of $\text{RECNUM} + L$ to see if it is greater than IMGROW. If so, the number of records input is just equal to $\text{IMGROW} - \text{RECNUM}$. Before these records are input, a flag is set (called EDFLAG) to signal that the last image row is to be input. These last image rows are then input, and the value of RECNUM is reset to 1. This is shown for an arbitrary choice

of the last two data rows in Figure 13 (m), (n), and (o). A flow diagram illustrating the row change operation is shown in Figure 21.

Interesting Point Interim Storage. In subroutine IOP1, after the simple variance has been computed for each window position by calling subroutine SVAR, this variance is tested against the user-defined threshold value to determine if an interesting point exists. If the calculated value of simple variance is greater than the user-defined threshold value, an interesting point exists at that window location, and subroutine IPSTOR is called to record the appropriate information. When IPSTOR is entered, the number of interesting points counter (IPKNT) is incremented, and the row and column locations of the interesting point are stored in arrays IPR and IPC respectively. To compute these locations, the value of WNDRWH is added to the value of ROWKNT to obtain the row location IPROW, and the value of WNDCLH is added to the value of COLKNT to obtain the column location IPCOL. The value of the interesting point is the value of simple variance obtained from subroutine SVAR, and is passed to IPSTOR as a parameter in the subroutine argument list as the variable VALUE. This variable is assigned to the array VALUIP.

In order to keep the number of interesting points from becoming excessively large, the array size for IPR, IPC, and VALUIP is restricted to M. If the value of IPKNT is greater than or equal to M after the interesting point information has been stored, a message that the interesting point storage limit has been exceeded is displayed to

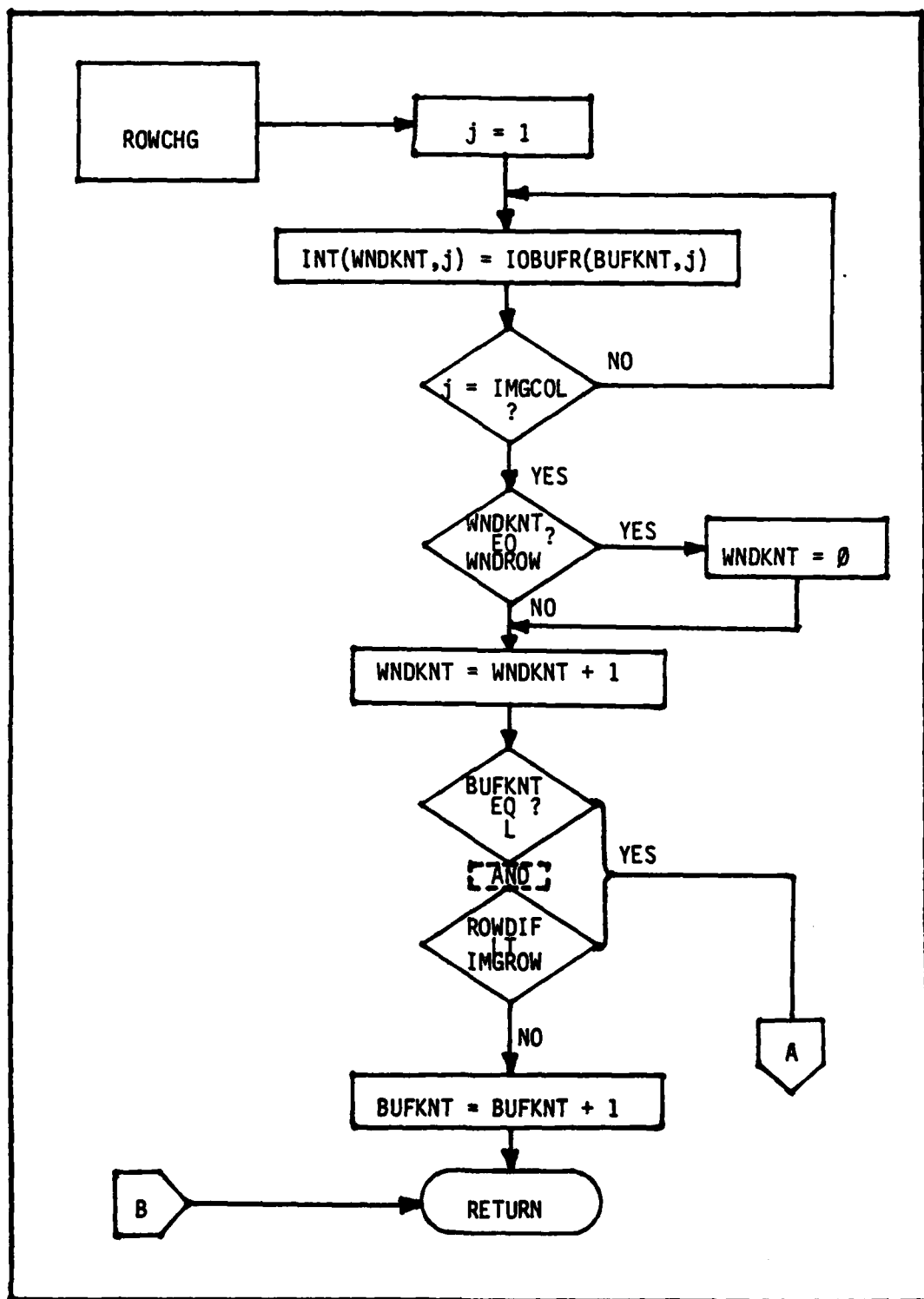


Figure 21. Row Change Flow Diagram

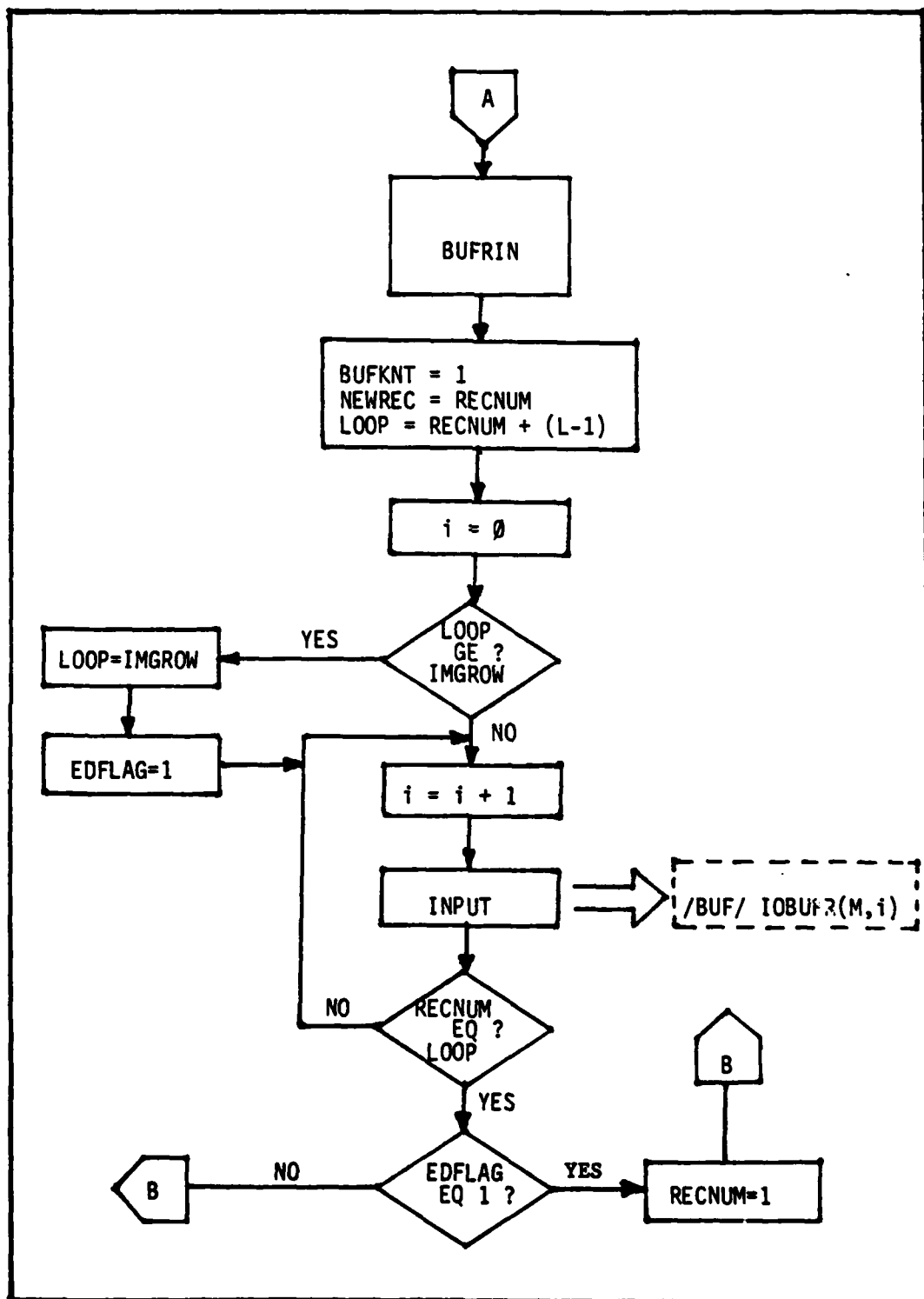


Figure 21. Row Change Flow Diagram

the user, and the row and column location of the center of the window where this occurred is printed on the line printer. Processing then terminates since no more interesting points can be stored.

Interesting Point Correlation Window. If the user has requested interesting point set storage in subroutine INTPTS, the value of IPFLAG is set equal to 1, as explained in the interesting point selection subsection of Chapter II. One of the items to be stored in the interesting point set is the correlation window associated with each interesting point. Thus, it is convenient to form this correlation window each time an interesting point is selected, and store this window on a temporary scratch file until called for permanent storage by subroutine IPSOUT.

Subroutine IPSTOR tests the value of IPFLAG after the interesting point information has been entered in arrays IPR, IPC, and VALUIP. If IPFLAG is equal to 1, IPSTOR calls subroutine CWDOUT to store the correlation window surrounding the interesting point on a temporary scratch file. The size of this correlation window is determined by the values of CWDROW and CWDCOL defined by the user, as previously explained.

Subroutine CWDOUT calls subroutine CORWND to determine the correlation window for the selected interesting point. CWDOUT then writes this correlation window (which is now located in the array IPWND) to the temporary scratch file using the logical unit number LUNCW previously defined. A flow diagram illustrating the method of

interesting point interim storage is shown in Figure 22.

Subroutine CORWND forms the correlation window about each interesting point by first calling subroutine CWDTST to determine the start and end locations of each window row and column. After this, subroutine INBUFR is called to input the necessary records from the image file to form the window. Byte-to-word number conversion is then performed. Finally, the window is broken up into sections and formed piece-by-piece. This is done because the edges of the correlation window may extend beyond the edges of the image for interesting points close to the edge of the image. In this case, the elements which extend beyond the edge of the image take on the value corresponding to the value on the image edge. These extended pixels are called "pseudo-pixels," and may be thought of as forming a continuous extension of the image boundary for the sole purpose of providing a means for totally and adequately defining the correlation window.

Subroutine CWDTST performs four tests. It first tests to see if the correlation window overlaps the left column of the image by setting the variable CWDLCL equal to the difference between the value of IPCOL and CWDCLH. If this difference is less than zero, the correlation window overlaps the left column of the image and the value of the correlation window column start position is set equal to $1 - \text{CWDLCL}$. If $\text{IPCOL} - \text{CWDCLH}$ is greater than or equal to zero, CWDCLS is set equal to 1. The same test is next performed for the row starting position using the variables CWDTRW and CWDROWS. The right hand column of the

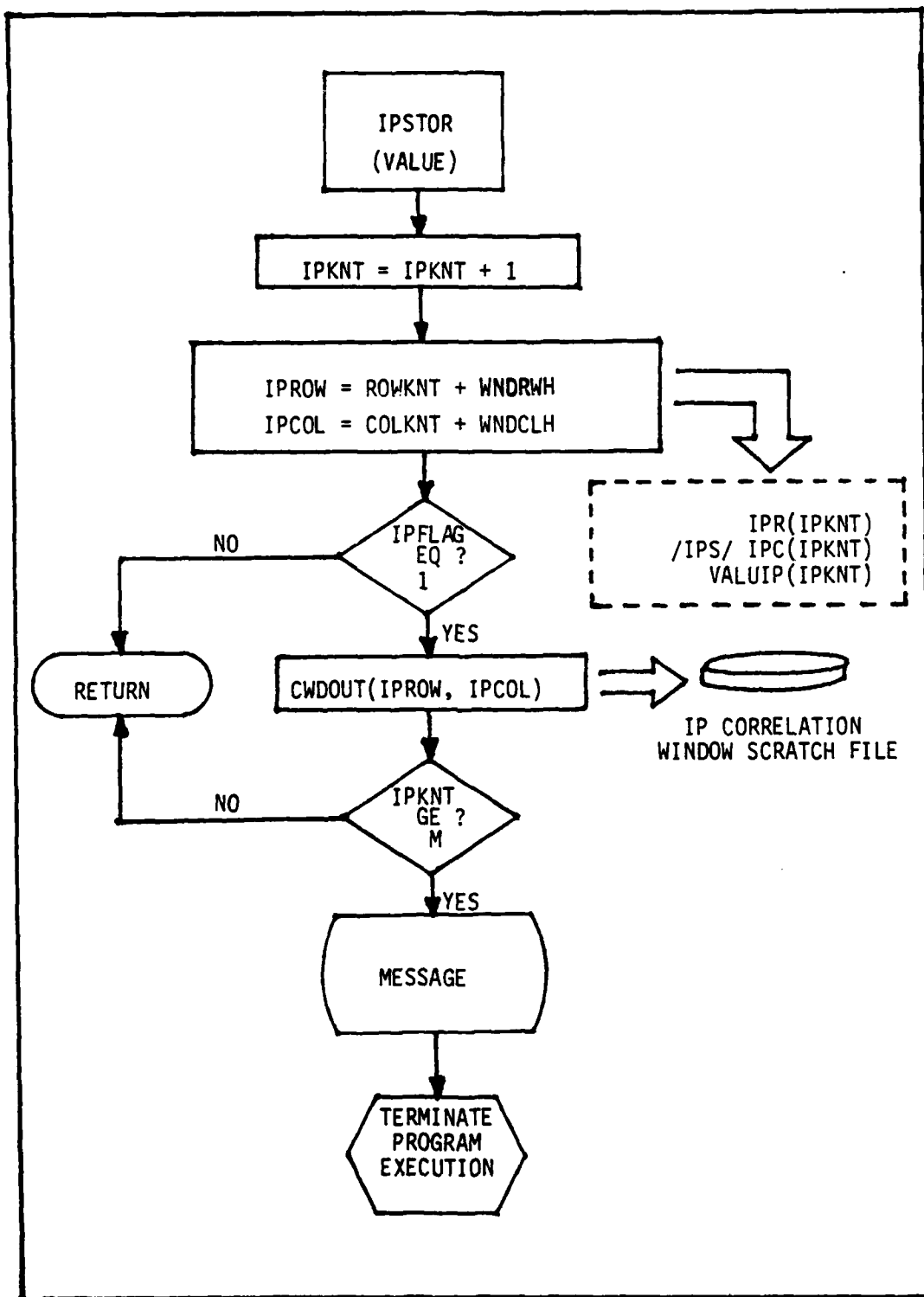


Figure 22. Interesting Point Interim Storage Flow Diagram

image is tested for by setting the variable CWDRCCL equal to $IMGCOL - (IPCOL + CWDCLH) + 1$. If this value is less than zero, the variable which determines the end of the correlation window columns, CWDCLE, is set equal to $CWDCOL + CWDRCCL$. If CWDRCCL is greater than or equal to zero, CWDCLE is set equal to CWDCOL. Finally, the bottom row of the image is tested for by setting CWDBRW equal to $IMGROW - (IPROW + CWDNRH) + 1$. If CWDBRW is less than zero, the variable which determines the end of the correlation window rows, CWDRE, is set to $CWDROW + CWDBRW$. If CWDBRW is greater than or equal to zero, CWDRE is set equal to 1 (not CWDROW as might be expected). The values of CWDROWS, CWDCLS, CWDRE, and CWDCLE are returned to subroutine CORWND to be used to form the correlation window.

After subroutine CORWND calls CWDTST, the record number of the first record of the image file used to form the correlation window is determined from the value of CWDROWS. The initial value of the correlation window starting element (position within the window of the first element to be formed) is given by the variable IPCKNT, and is also determined by the value of CWDROWS. The relative location of the image data in the records input to form the correlation window is defined by the variable IMCLST. After these variables are defined, image data from the image file is input into a storage buffer when subroutine INBUFR is called.

Subroutine INBUFR reads the record from the image file associated with the value of RECNUM defined in subroutine CORWND. Note that this

value of RECNUM is totally independent of the variable RECNUM being used by the interest operator, and it is not stored in a COMMON area. When the correlation window is formed, and control is passed back to the interest operator program, this value of RECNUM is lost, and the value of RECNUM stored in the area COMMON /COUNT/ is used to determine records read thereafter. This is perfectly permissible when using direct access files, and this is the reason that this type of file access was chosen. Subroutine INBUFR stores the input record in the array IBUF(M) in the area COMMON /IBUF/.

The correlation window is formed one row at a time, and each row is placed on its proper location in array IPCWND after it is formed. The rows are placed end-to-end with the interesting point being located at the center element of array IPCWND. In subroutine CORWND, after byte-to-word number conversion has been performed, correlation window rows are formed depending on the relative location of the correlation window with respect to the image. The window is then formed in sections depending upon this relative location.

If the value of CWDCLS returned by CWDTST in subroutine CORWND is greater than 1, this implies that the leftmost column of the correlation window overlaps the leftmost column of the image. The amount of this overlap is CWDCLS-1. After byte-to-word number conversion is performed, each record input into the byte buffer array IBUF is placed in the integer word array INTBUF. Thus, if CWDCLS is greater than 1, the first CWDCLS-1 elements of the correlation window

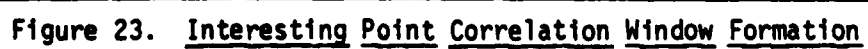
row corresponding to the input record are given the value of INTBUF(1). The window row is completed by copying the elements starting from INTBUF(1) and ending with the array position corresponding to the value of CWDCLC into the array IPCWND. However, if CWDCLC extends the correlation window row beyond the rightmost column of the image, the following CWDCOL - (CWDCLC + 1) row elements are given the value INTBUF(IMGCOL). Thus, in theory, each correlation window row could begin before the leftmost column of the image, and extend beyond the rightmost column, and still be satisfactorily filled (although this situation is not likely to occur in normal processing with large images.)

In like manner, if the value of CWDROWS is greater than 1, this implies that the uppermost row of the correlation window overlaps (starts prior to) the uppermost row of the image. The amount of this overlap is CWDROWS-1. Thus, if CWDROWS is greater than 1, the first CWDROWS-1 rows of the correlation window are assigned the same values as the correlation window row corresponding to the first (uppermost) image row. Recall that this row may also overlap the image to the left. In this case, the entire upper left-hand corner of the correlation window which overlaps the image will contain the same value as the upper left-hand corner element of the image. This expansion of the upper left-hand corner element of the image into the upper-left hand portion of the correlation window is called "fanning." The upper right-hand element, and lower left- and right-hand elements of the image may also be fanned if the appropriate overlap occurs.

Finally, if CWDWE is greater than 1, and RECNUM is equal to IMGROW, this implies that the bottom row of the correlation window is below the bottom row of the image. Thus, if CWDWE > 1 and RECNUM = IMGROW, the last CWDROW - (CWDWE + 1) rows of the correlation window are assigned the same values as the correlation window row corresponding to the last image row (IMGROW). Again, these last correlation window rows may overlap the image to the left or right such that fanning will occur.

Therefore, in a hypothetical sense, the correlation window could overlap the image on the left, right, above, and below simultaneously (i.e., the entire image enclosed by the correlation window) and still be adequately formed by subroutine CORWND. In this case, the interesting point would be at the center of the image, assuming that the overlap is the same on all sides. This case is shown as a way of summarizing all the possible overlap conditions in Figure 23. The arrows indicate the values that the pseudo-pixels take on from the corresponding image pixels.

Therefore, to select interesting points using the simple variance interest operator (IOPI), a variance threshold is first defined by the user, and the appropriate parameters are initialized. The simple variance is then computed and tested against the user - defined threshold. If the value of simple variance computed is greater than the threshold value, an interesting point exists at the center of the window at that location. The row and column location of the



interesting point are then stored along with its value. If interesting point set storage has been requested, the correlation window surrounding the interesting point is formed and stored on a temporary scratch file. Next, the processing window is advanced sequentially to the next image position. When this window reaches the bottom right-hand corner of the image, processing terminates and control is passed back to the interesting point selection control program. A flow diagram illustrating the method of interesting point selection using the simple variance interest operator is shown in Figure 24.

Directed Variance Interest Operator

Again, consider the $n \times m$ dimensional array of numbers, A . The directed variance is the minimum root-mean-square (RMS) directed difference over the array area [Ref. 9: (3-26)]. To compute the directed variance, sums of the squares of the differences of the elements of A are formed in four directions. These four directional variances are expressed as

$$D_1 = \frac{1}{nm} \left\{ \sum_{i=1}^n \sum_{j=1}^m (a_{ij} - a_{i,j+1})^2 \right\} \quad (13)$$

$$D_2 = \frac{1}{nm} \left\{ \sum_{i=1}^n \sum_{j=1}^m (a_{ij} - a_{i+1,j})^2 \right\} \quad (14)$$

$$D_3 = \frac{1}{nm} \left\{ \sum_{i=1}^n \sum_{j=1}^m (a_{ij} - a_{i+1,j+1})^2 \right\} \quad (15)$$

$$D_4 = \frac{1}{nm} \left\{ \sum_{i=1}^n \sum_{j=1}^m (a_{i+1,j} - a_{i,j+1})^2 \right\} \quad (16)$$

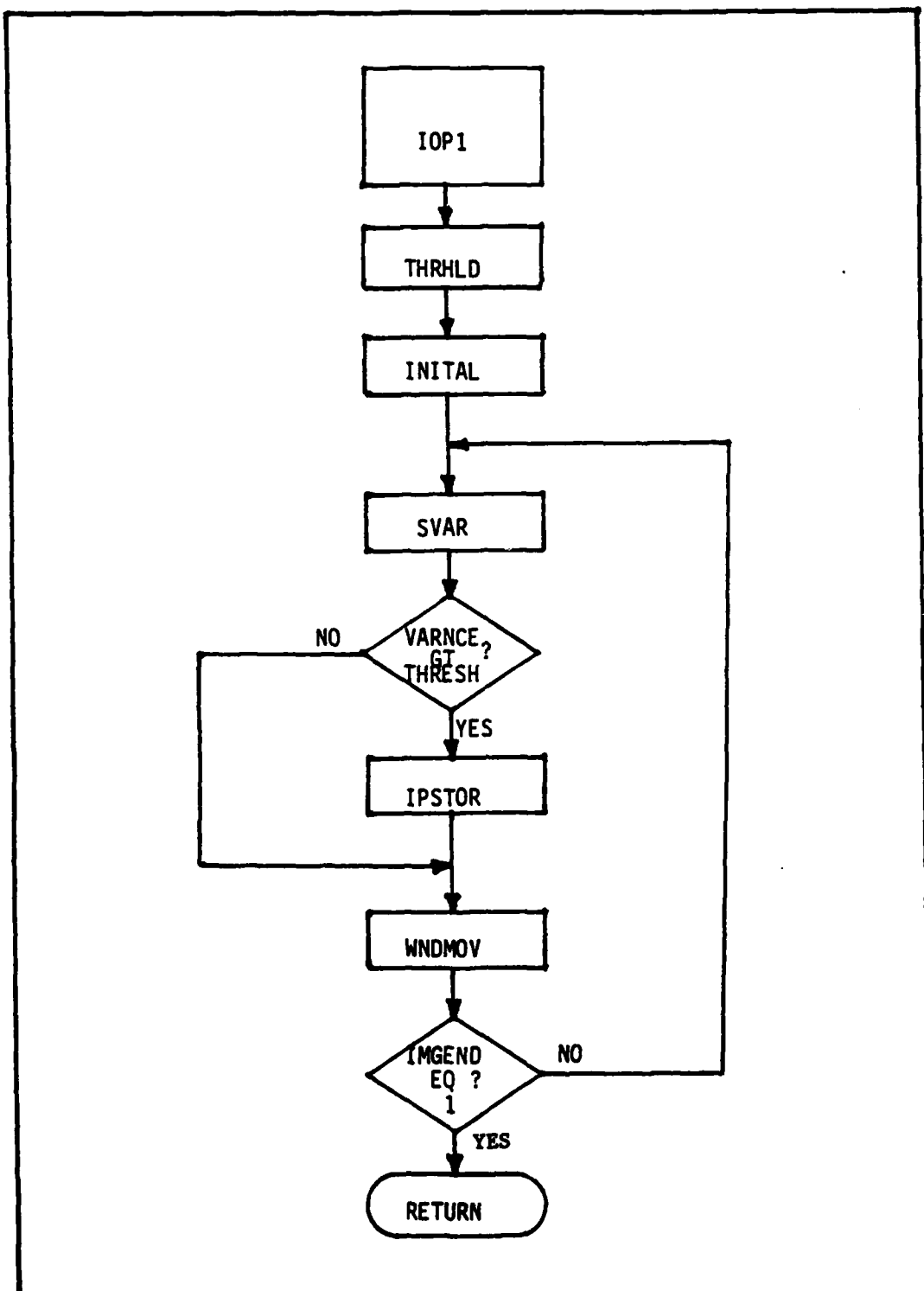
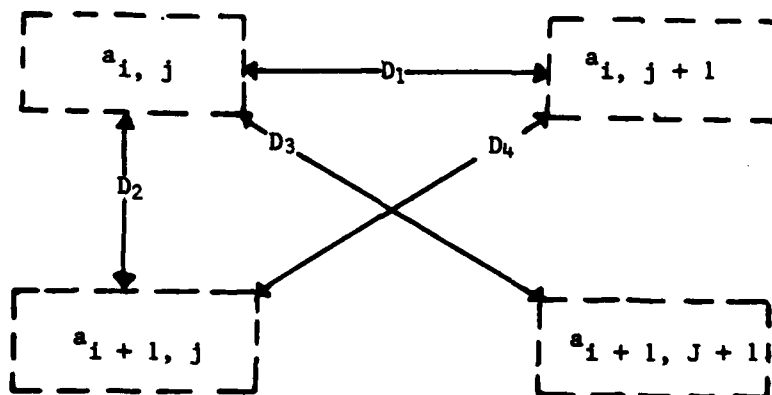


Figure 24. Simple Variance Interest Operator Flow Diagram

In the above equations, D_1 is the directional variance of the sum of the squares of the differences of each row of A, D_2 is the directional variance of the sum of the squares of the differences of each column of A, D_3 is the directional variance of the sum of the squares of the differences of the upper-to-lower diagonal elements of A, and D_4 is the directional variance of the sum of the squares of the differences of the lower-to-upper diagonal elements of A. This is shown below for an arbitrary four-element array location:



The overall directed variance, D , is defined to be the minimum of these four directional variances. Thus

$$D = \text{MIN} (D_1, D_2, D_3, D_4) \quad (17)$$

Areas with poor visual texture will have a low directed variance because adjacent samples differ little in any of the four directions. Also, areas with linear edges will exhibit low directional variances in the direction of the edge. Conversely, areas with high directed variances should avoid these defects. Thus, interesting points are defined to be local maxima in this directed variance interest operator [Ref. 9: (3-4)].

In program DIDA, to select interesting points using the directed variance interest operator, the user enters a 2 when queried concerning the desired interest operator in subroutine IOPSEL, as explained in the interesting point selection section of Chapter II above. This entry causes IOPSEL to call subroutine IOP2 which computes the directed variance of each processing window position, and then determines if an interesting point exists. If an interesting point is found to exist at a certain window location, it is placed in arrays IPR, IPC, and VALUIP in the area COMMON /IPS/. If interesting point set storage has been requested, the correlation window corresponding to the interesting point is stored on a temporary scratch file, as previously explained.

To begin with, subroutine IOP2 calls subroutine THRHLN to allow the user to specify the variance threshold to be used to test for interesting points. Next, subroutine EPSFND is called to allow the user to interactively specify a small threshold value (given by the variable EPSLON) to be used for the maximum directed variance tests

(explained later). As usual, entry checking and correction are provided. After EPSFND is called, subroutine INITAL is called to initialize various parameters associated with interesting point processing.

To compute the directed variance for each window position, IOP2 next calls subroutine DVAR. This subroutine stores the computed values in the directed variance storage array D(2,M) by calling subroutine DVSTOR. The directed variance values are stored in the first row of array D for the initial pass of the window across the image (Figure 12 (a), (b), and (c)). Thereafter (i.e., for the second and subsequent window passes), these values are stored in the second row of array D. The number of directed variance values stored in one row of array D is determined by the variable DCOL, which is a function IMGCOL and WNDCOL. Subroutine DVAR plays a role similar to that of subroutine MEANFD in the simple variance interest operator described above. The variables DCOL and EPSLON, and the array D(2,M), are stored in the area COMMON /DSTOR/.

After each value of directed variance is computed and stored in array D, subroutine WNDMOV is called to advance the processing window one image column (or row) position. When the first row of array D is filled (i.e., the processing window reaches its rightmost position on the image), subroutine MAXDVR is called. This subroutine determines the local maxima of the directed variance values by comparing these values with their adjacent neighbors in array D. After the initial

window pass across the image, only adjacent neighbors in one row (the first row) are compared. After the second and subsequent passes of the processing window, adjacent neighbors of both D rows are compared. The directed variance value which is greater than all its adjacent neighbors is left positive, and all the neighbors are set negative. Thus, after two passes of the window across the image, the directed variances which remain positive in the first row of array D are the local maxima for the first pass of the window across the image.

Since two passes of the window across the image are required before interesting points can be selected, there is always a phase lag of one window pass across the image (i.e., one row) in the processing cycle. After the second window pass, subroutine IOP2 tests the first row of directed variance values in array D for magnitudes greater than the user-defined threshold, THRESH. Note that all the negative values will never pass this test; only the ones remaining positive which are greater than THRESH (i.e., the local maxima) will pass. When a directed variance value greater than THRESH is found, this is an interesting point. Subroutine IPSTOR is then called to store this interesting point, as explained previously. However, before IPSTOR can be called, ROWKNT must be reduced in value by 1 because of the cyclic phase lag. The value of COLKNT is set equal to the corresponding D column because the window is in the same column location even though there is a one row phase lag. After the interesting point has been stored, ROWKNT is incremented by 1 again, and COLKNT is returned to its value of DCOL (which it had achieved just prior to the search of row 1

of array D for interesting points). This same procedure is repeated for each element of the first row of array D until the last element is checked.

When the search of the first row of array D for interesting points is complete, subroutine DRWCHG is called. This subroutine replaces the contents of the first row of array D with the contents of the second row. Subroutine WNDMOV is then called which advances the window to the next image row for processing. As the window is passed across the image again, the computed directed variance values are placed in the second row of array D. When the window reaches its rightmost position on the image, subroutine MAXDVR is called to determine the local maxima of the directed variance values in the first row based on the new second row values. This first D row is then searched for the local maxima above THRESH, as before. This process continues until the end of the image is reached (IMGEND = 1).

Recall that there is a one row phase lag involved in the processing cycle. Thus, when the processing window reaches the end of the image, one row still remains to be searched for interesting points. These will now be the local maxima above THRESH contained in the second row of array D. Thus, when WNDMOV signals IOP2 that the end of the image has been attained, the second row of array D is searched for interesting points, and interesting point processing is complete. Finally, subroutine CLOSKP is called to close the image file.

Directed Variance Computation. To compute the directed variance,

subroutine IOP2 calls subroutine DVAR to compute the four directional variances as given in equations (13), (14), (15), and (16), and the overall directed variance as given in equation (17). The sums of the squares of the differences of the intensity elements for each window position are stored in the area COMMON /DRVAR/ by the variables XSUM, YSUM, ASUM, and BSUM. These sums are formed by recursive updates the same way COLSUM and SQRSUM were formed in subroutine MEANFD, as previously explained.

To form these sums, when the processing window is in its rightmost position on the image, before being passed across, subroutine DVAR calls subroutine ADDDIF. This subroutine forms partial sums of the squares of the differences of the intensity elements in four directions for each column of the image. Each of these columns is of length WNDROW. Thus, the function of subroutine ADDDIF is identical to that of subroutine ADDCOL mentioned above (also note equations (6) and (9)).

In subroutine ADDDIF, the sums of the squares of the differences of adjacent elements for each column of the image are stored in the area COMMON /REG2/ in the arrays X(M), Y(M), A(M), and B(M). The elemental intensity differences are formed by

$$\text{DIFROW} = \text{INT}(i, j) - \text{INT}(i, j+1) \quad (18)$$

$$\text{DIFCOL} = \text{INT}(i, j) - \text{INT}(i+1, j) \quad (19)$$

$$\text{DIFULD} = \text{INT}(i, j) - \text{INT}(i+1, j+1) \quad (20)$$

$$\text{DIFLUD} = \text{INT}(i+1, j) - \text{INT}(i, j+1) \quad (21)$$

The squares of these intensity differences are then placed in the variables DFSQRW, DFSQCL, DFSQUL, and DFSQLU, and the four column sums

are formed from these squares.

Note that for the last image column position, no operation is performed. In other words, if the last column is left unaccounted for, there will be one less partial sum than the number of image columns. To account for this extra position, the values of $X(IMGCOL)$, $A(IMGCOL)$, and $B(IMGCOL)$ are merely assigned the values $X(IMGCOL-1)$, $A(IMGCOL-1)$, and $B(IMGCOL-1)$ respectively. This is nothing more than taking the previous squared differences in reverse, which yield the same value. Thus, they are equated. To determine the last squared column difference, the difference value is formed by

$$DFCLED = INT(i, IMGCOL) - INT(i+1, IMGCOL) \quad (22)$$

The square of this value is placed in the variable $DFSQCE$. This variable is then used to form the value of $Y(IMGCOL)$. Having formed the values of all the partial sums, $ADDDIF$ passes control back to subroutine $DVAR$.

In subroutine $DVAR$, each time the window is returned to its leftmost position on the image, the initial values of $XSUM$, $YSUM$, $ASUM$, and $BSUM$ are formed by adding up $WDCOL$ partial sums. As the window then moves across the image, these initial values are recursively updated according to the equations

$$XSUM = XSUM + X(COLDIF) - X(COLKNT-1) \quad (23)$$

$$YSUM = YSUM + Y(COLDIF) - Y(COLKNT-1) \quad (24)$$

$$ASUM = ASUM + A(COLDIF) - A(COLKNT-1) \quad (25)$$

$$BSUM = BSUM + B(COLDIF) - B(COLKNT-1) \quad (26)$$

The four directional variances DVAR1, DVAR2, DVAR3, and DVAR4 are then computed by the product of WNDINV and the corresponding value of XSUM, YSUM, ASUM, and BSUM. Therefore, the final value of overall directed variance for each window position is given by

$$\text{DIRVAR} = \text{MIN}(\text{DVAR1}, \text{DVAR2}, \text{DVAR3}, \text{DVAR4}) \quad (27)$$

As in the case of subroutine MEANFD, each time the window returns to its leftmost position on the image after the first row has been processed, subroutine ROWCHG is called to interchange the corresponding intensity array and input buffer rows (see Figure 21).

Directed Variance Local Maxima. Once a row of directed variance values have been stored in array D, subroutine MAXDVR is called. This subroutine is divided into two main sections: (1) test for local maxima of the first row of directed variance values; and (2) tests for local maxima using both rows of directed variance values after the initial row has been tested. In the single row test, each element is compared with its adjacent neighbor as the row is scanned from left to right. The dual row test is broken up into three parts: (1) comparison of the left-end three elements of array D; (2) comparison of the five middle elements of array D as the array is scanned from left to right; and (3) comparison of the right-end four elements of array D.

In the single row tests, as the row is scanned, the element to the right is subtracted from the absolute value of the preceeding element.

If this difference is greater than or equal to the user-defined threshold, EPSLON, this implies that the left-hand element is larger than the right-hand element. Thus, the right-hand element is set negative and the test proceeds to the next sequential element pair. Note that the absolute value of the right-hand element does not have to be taken since this element will always be positive prior to comparison. However, the left-hand element could have been set negative by the previous test. Thus, its absolute value must be compared.

On the other hand, if the above test difference is less than EPSLON, this implies that the element on the right is larger than the element on the left. In this case, this element is tested to see if it has already been set negative. If so, no action is taken. If not, it is set negative, and the test proceeds to the next sequential element pair.

In the dual row tests, two forms of comparison are used. The first form is to compare the difference of the two elements being tested to see if this difference is greater than or equal to the user-specified value of EPSLON. The second form is to compare the elemental difference to see if it is strictly greater than EPSLON. These two comparison test forms are applied alternately on each successive call to subroutine MAXDVR from IOP2. In other words on an arbitrary call to MAXDVR sometime after processing has begun, if the greater than or equal to test is applied, then on the next call to

MAXDVR, the strictly greater than test is applied. This alternating comparison sequence continues until the last image row has been processed.

The reason for alternating between greater than or equal to and strictly greater than comparison test forms is to treat values of directed variance which are local maxima and close to each other in absolute value. Consider the test of two neighboring elements of array D. If the difference of the absolute values of these elements is greater than EPSLON, the smaller element is set negative. Now assume that the larger of these two elements is a local maximum. In this case, it will emerge non-negative after completion of the test sequence. If the greater than or equal to test is being applied, the same element will emerge non-negative for an elemental difference equal to EPSLON as well as for an elemental difference greater than EPSLON.

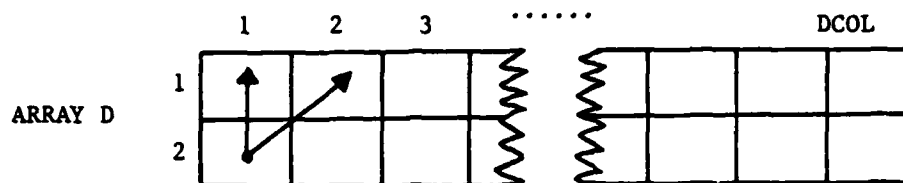
This selection of the same non-negative element for the case where the elemental difference is either greater than or equal to EPSLON introduces a bias into the placement of the interesting points on the image being processed. Consider the case when EPSLON is equal to zero. In this case, if the elemental difference is zero, both elements are equal, but the element set negative is the same as if the test result had been greater than zero. There is no reason to select this element as being negative over the other one if both elements are equal (and are local maxima).

Now consider the strictly greater than comparison test. For the

case of EPSLON equal to zero, if the elemental difference is zero, the element opposite the one set negative in the greater than or equal to test is set negative. If both elements are locally maximal, then this opposite element, rather than the other one, will also emerge non-negative at the completion of the test sequence. Therefore, by alternating between the two comparison test forms, the bias introduced by using only the greater than or equal to test is removed.

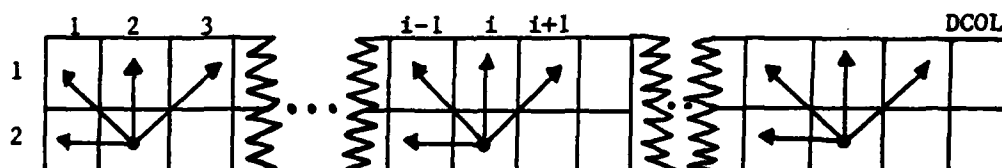
The magnitude of EPSLON is generally kept small. For the tests performed in this effort, a value of EPSLON equal to 1 was used. The value of EPSLON determines the amount of difference in absolute values of the elemental neighbors of array D the alternating test sequence will allow before switching the element that gets set negative. As mentioned above, when EPSLON is zero, this difference is also zero. For EPSLON equal to 1, the two neighboring elements of D can differ in absolute value by 1 and the same element will still be set negative in both comparison forms. However, if this difference exceeds 1, then the element opposite the one set negative in the greater than or equal to test will be set negative in the strictly greater than test. Allowing values of EPSLON other than zero gives the user a certain amount of latitude in specifying how close two variance values must become before being treated as the same value for purposes of comparison testing.

To begin the dual row tests after MAXDVR is called, the left-end three elements of array D are compared as shown below:



In this case, the absolute value of element $D(1,1)$ is first compared with element $D(2,1)$ using one of the comparison test forms. It is not necessary to compare the absolute value of $D(2,1)$ here since it cannot be negative. Next, the absolute value of $D(1,2)$ is compared with the absolute value of $D(2,1)$. In this case, the absolute value of $D(2,1)$ must be compared because $D(2,1)$ could have been set negative in the first test.

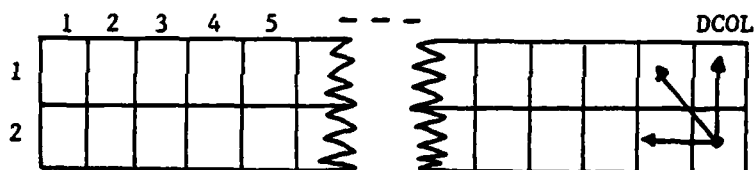
After the three left-end elements have been tested, the intermediate mid-row elements of array D are tested five at a time as the array is scanned from left to right as shown below:



For the i -th comparison test, $D(2,i)$ is first compared with the absolute value of $D(1,i)$. Note here that if $D(2,i)$ is greater than or equal to (or strictly greater than depending upon which test form is being used) the absolute value (ABS) of $D(1,i)$, and $D(1,i)$ itself is

positive, then the test need not proceed any further. $D(1,i)$ positive means it is larger than $D(1,i-1)$ and $D(2,i-1)$ from the previous test. Thus, since $D(2,i)$ is larger than $D(1,i)$, it is also larger than these other values. Since $D(1,i+1)$ will be compared on the next test, the scan can proceed to the $(i+1)$ -th comparison test. Other such simplifications are performed, where appropriate, to reduce the amount of time required for each portion of the test sequence (and thus reduce the total amount of processing time). However, in the worst case, all four comparisons of the five elements must be performed.

To complete the dual row tests, the right-end four elements of array D are compared as shown below:



As in the case of the intermediate row tests, $D(2,DCOL)$ is first compared with $ABS(D(1,DCOL))$. If $D(2,DCOL)$ is greater than (or greater than or equal to) $ABS(D(1,DCOL))$, and $D(1,DCOL)$ is positive, the test ends. Note here, as before, that $D(1,DCOL)$ positive implies that it is larger than $D(1,DCOL-1)$ and $D(2,DCOL-1)$. Thus, if $D(2,DCOL)$ is larger than $ABS(D(1,DCOL))$, and $D(1,DCOL)$ is positive, then $D(2,DCOL)$ is also larger than $D(1,DCOL-1)$ and $D(2,DCOL-1)$, and testing need not proceed any further.

After the completion of the right-end element comparisons, the dual row test sequence is complete, and control passes from subroutine MAXDVR to IOP2. The elements remaining positive in the first row of array D will be the local maxima upon which interesting point selection is based. If the debug version of program DIDA is being executed, both rows of array D are printed on the line printer after the test sequence is complete.

Therefore, to select interesting points using the directed variance interest operator (IOP2), the variance threshold and value of EPSILON are first defined by the user, and the appropriate parameters are initialized. The directed variance value for each window position is then computed and stored. After one complete pass of the window across the image, the local maxima of the stored directed variance values are determined. After two complete passes of the window, these local maxima are tested against the user-defined variance threshold to determine if an interesting point exists. If so, subroutine IPSTOR is called to store the interesting point (along with its associated correlation window, if requested). When the processing window reaches the lower right-hand corner of the image, the last row of directed variance values is tested for interesting points and processing terminates. A flow diagram illustrating the method of interesting point selection using the directed variance interest operator is shown in Figure 25.

Edged Variance Interest Operator

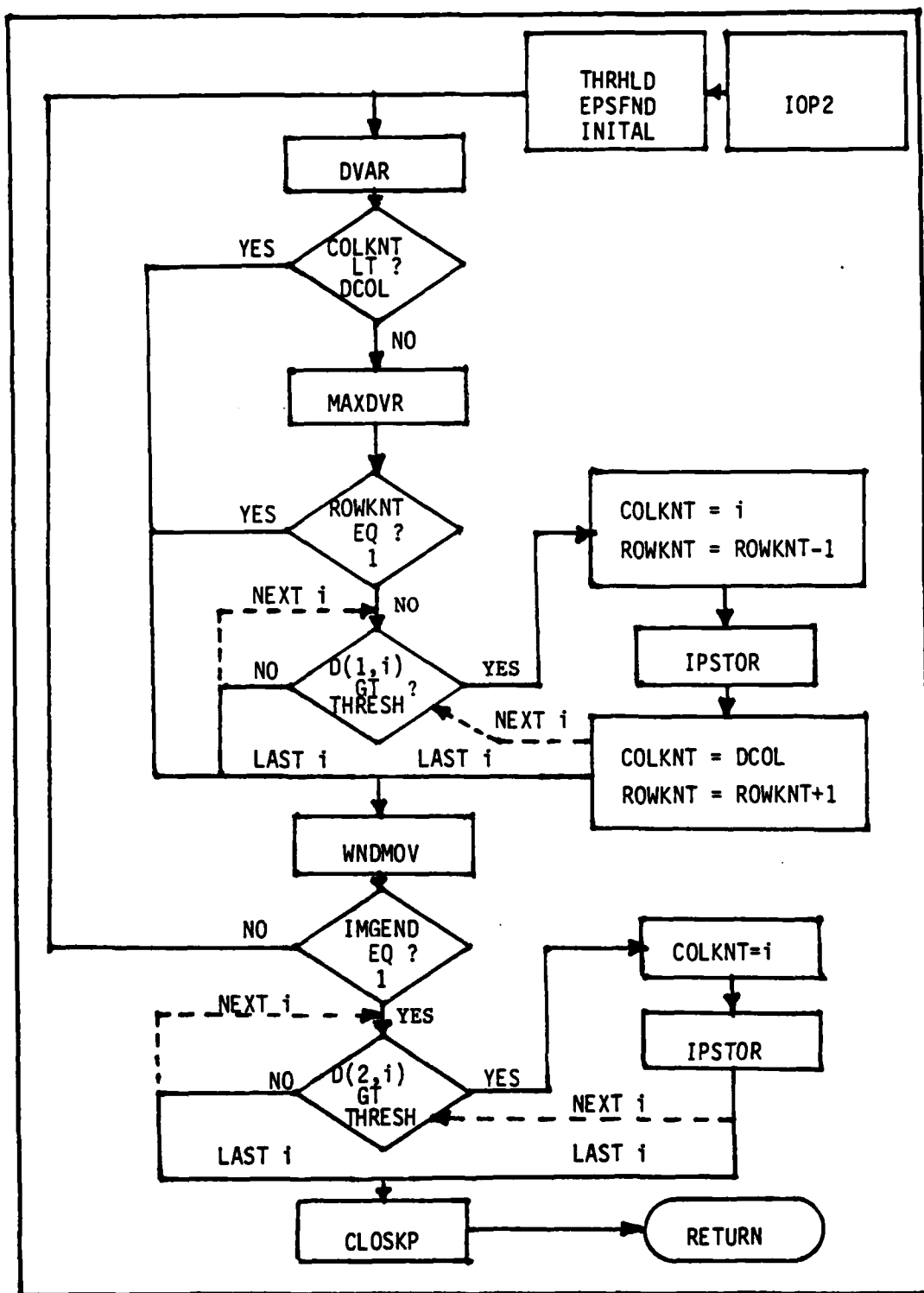


Figure 25. Directed Variance Interest Operator Flow Diagram

Edged variance is a function of both simple variance and directed variance. To compute the edged variance over a given window position, the four directional variances DVAR1, DVAR2, DVAR3, and DVAR4 are computed as explained in the directed variance interest operator section above. Four perpendicular ratios of these directional variances are then formed as a measure for determining the presence of an edge within the window area. This measure takes advantage of the fact that a window with a strong edge within it will have a much greater information content across the edge than along it. The minimum of these four ratios provides a measure of the relative strength of the information parallel to the dominant edge within the window area, independent of the contrast of the image. Since this measure does not give any indication of the amount of information within the window, it is multiplied by the simple statistical variance to produce the edged variance. The method of computing the simple variance is described in the simple variance interest operator section above. Thus, the edged variance, E, is given by [Ref. 9: (3-5)]

$$E = V \left\{ \text{MIN} \left[\frac{D1}{D2}, \frac{D2}{D1}, \frac{D3}{D4}, \frac{D4}{D3} \right] \right\} \quad (28)$$

In the above equation, V is the simple statistical variance computed by equations (1) and (5). D1, D2, D3, and D4 are the four directional variances given by equations (13), (14), (15), and (16) respectively.

In program DIDA, to select interesting points using the edged variance operator, the user enters a 3 when queried concerning the

desired interest operator in subroutine IOPSEL, as explained in the interesting point selection section of Chapter II above. This entry causes IOPSEL to call subroutine IOP3 which computes the edged variance of each window position, and then determines if an interesting point exists. If an interesting point is found to exist at a certain window location, it is placed in arrays IPR, IPC, and VALUIP in the area COMMON /IPS/. If interesting point set storage has been requested, the correlation window corresponding to the interesting point is stored on a temporary scratch file, as previously explained.

To begin with, subroutine IOP3 calls subroutine THRHLDD to allow the user to specify the variance threshold to be used to test for interesting points. Next, subroutine INITAL is called to initialize various parameters associated with interesting point processing. To select the interesting points, subroutine EVAR is called which returns the value of edged variance for a given window position. This value of variance (specified by the variable EDGVAR) is then tested against the user-defined threshold. If the value of EDGVAR is greater than the value of THRESH, an interesting point exists at that window location, and subroutine IPSTOR is called to store the location and value of the interesting point (and the correlation window, if requested). Finally, subroutine WNDMOV is called to advance the window to the next sequential row and column position (see Figure 12). If the last window position is detected by subroutine WNDMOV (Figure 12(1)), interesting point processing is complete and subroutine CLOSKP is called to close the image file.

Edged Variance Computation. To compute the edged variance at each window position, subroutine EVAR first calls subroutine SVAR to compute the simple variance. Next, subroutine DVAR is called to compute the four directional variances DVAR1, DVAR2, DVAR3, and DVAR4. When subroutine EVAR is entered, the edged variance flag EFLAG is set equal to 1 to signal subroutine DVAR that the edged variance, rather than the directed variance, is being computed. After the four directional variance values have been computed, subroutine RATIO is called to calculate the four perpendicular ratios of these values according to equation (28).

Subroutine RATIO receives the values of the four directional variances from subroutine DVAR and returns the values of the four perpendicular ratios according to the following equations.

$$\begin{array}{lll} \text{RATIO1} & = & \text{DVAR1/DVAR2} & (29) \\ \text{RATIO2} & = & \text{DVAR2/DVAR1} & (30) \\ \text{RATIO3} & = & \text{DVAR3/DVAR4} & (31) \\ \text{RATIO4} & = & \text{DVAR4/DVAR3} & (32) \end{array}$$

This subroutine also tests the divisor in each equation for a zero value prior to division. If it is zero, a large number is assigned to that ratio. If both numerator and denominator are zero, the ratio is set to zero.

After subroutine RATIO returns the values of the above four ratios to subroutine DVAR, the minimum of these ratios is obtained by

$$\text{RTOMIN} = \text{MIN}(\text{RATIO1}, \text{RATIO2}, \text{RATIO3}, \text{RATIO4}) \quad (33)$$

This value of the minimum of the four ratios, along with the value of EFLAG is stored in the area COMMON /EDVAR/. Thus, when subroutine DVAR returns control to subroutine EVAR, the value of edged variance is computed by the simple expression

$$\text{EDGVAR} = \text{VARNCE} * \text{RTOMIN} \quad (34)$$

Therefore, to select interesting points using the edged variance interest operator (IOP3), a variance threshold is first defined by the user, and the appropriate parameters are initialized. The edged variance is then computed and tested against the user-defined threshold. If the value of edged variance computed is greater than the threshold value, an interesting point exists at the center of the window at that location. The row and column location of the interesting point are then stored along with its value. If interesting point set storage has been requested, the correlation window surrounding the interesting point is formed and stored on a temporary scratch file. Next, the processing window is advanced sequentially to the next image position. When this window reaches the bottom right-hand corner of the image, processing terminates and control is passed back to the interesting point selection control program. The flow diagram illustrating the method of interesting point selection using the edged variance interest operator is identical to the flow diagram of the simple variance interest operator shown in Figure 24. The only difference is that subroutine EVAR is called after subroutine INITIAL rather than subroutine SVAR.

Interest Operator Evaluation

Each of the above-mentioned interest operators functions differently depending upon the specific application in which it is being used. Simple statistical variance has a low value in areas of low information content. The directed variances will also be small in areas of low information content. However, if the area has a strong linear edge with little information on either side, then the directional variance which most nearly parallels the edge will be small. By defining the interest measure to be the maximum of the minimums of the directed variances, areas with low information content, and even uni-directional information, can be rejected. Thus, simple variance and directed interest operators find points along the strong irregular edges which mark tree/grass boundaries, corners of buildings, object outlines, and other such image features, and ignore open areas having more subtle features. On the other hand, edged variance gives a combination of strong and subtle features, while avoiding excessively plain areas.

On the basis of pure performance, edged variance should provide the best interest measure. However, it is computationally the most expensive, requiring the calculation of six sums (five of which are squared quantities), five divides, a MINIMUM operation, and seven multiplies. Directed variance is less expensive, requiring four sums (all of squared quantities), a divide, a MINIMUM, and four multiplies. Simple statistical variance is the cheapest, requiring only two sums

(of which only one is a square), two multiplies, and a divide.

If the image to be processed contains mostly natural terrain, the edge rejection properties of directed variance and edged variance are not needed. In this case, the cheaper simple variance interest operator should be used. If the images contain many linear features, such as roads, houses, and so forth, then strictly linear edge rejection is important. In these cases, the directed variance interest operator is the preferred choice. In images which contain relatively featureless terrain, the increased performance of edged variance is needed. However, if the terrain provides fairly contrasty images, the cheaper directed variance operator may be employed [Ref. 9: (3-26)-(3-29)].

IV. Interesting Point Matching Algorithms

Without citing many examples, the applications of CATVI are many and varied. Of these, tracking is perhaps the most obvious. Two types of trackers can be distinguished: (1) background trackers; and (2) target trackers. A background tracker identifies an object in the background of an image (a house, for example), and monitors the location of that object as the imaging sensor moves past it. Target tracking on the other hand, occurs when the movement of an object relative to the background must be monitored [Ref. 10: 7]. As an example of this, consider the moving enemy tank mentioned on page 1 of the introduction section. In order for the airborne munitions platform to destroy this tank, it must be able to track the tank as it moves, while preparing to discharge weaponry against it. As the imaging sensor of the platform receives sequential images of the target area, the movement of the tank within these images creates a disparity from one image to the next, as mentioned at the beginning of Chapter III. The first question which might arise then is how the movement of the tank can be tracked from image-to-image.

One method of accomplishing this tracking is to use disparity analysis. Using an interesting point selection algorithm, interesting points (in this case, points on the tank) can be located in each image. However, merely locating the interesting points in each image does not provide sufficient information to track the tank. In order to track it, the interesting points on the tank in one image must be identified

in the next image as being the same point. This process of identifying the same interesting points from image-to-image is called "matching."

Matching is a natural way to approach disparity analysis [Ref. 4: 333]. Dougherty accomplishes this matching by similarity detection which minimizes the norm of the difference between two images [Ref. 17: 31]. A method which assigns probabilities to each potential candidate match point using the method of relaxation-labeling is suggested by Barnard and Thompson [Ref. 4: 335-336]. This is the method discussed below. In still another approach, Hannah uses reduction matching accomplished by maximizing the normalized cross-correlation over small windows surrounding the interesting points [Ref. 12: 203]. In this case, a simple spiraling grid search is used to obtain a precise match [Ref. 9: (3-5)].

Interesting point matching is the very heart of disparity analysis since it forms the basis for the extraction of all pertinent information from image-to-image. During this thesis effort, matching was accomplished by successive approximation under a probabilistic relaxation-labeling scheme as described by Thompson and Barnard [Ref. 18: 24-25] and more generally by Zucker, LeClerc, and Mohammed [Ref. 19: 117-123]. The relaxation-labeling method uses iterative probabilistic classification (matching) rather than making firm classification decisions immediately. Classification probabilities at a given iteration are allowed to depend on decisions made at the previous iteration. The probability updating at a given iteration is

done in parallel over the entire image [Ref. 20: 79].

In the studies discussed above, three properties of image pairs which can strongly influence the matching process can be identified: (1) discreteness; (2) similarity; and (3) consistency. Discreteness is a measurement of the individual distinctness of a point, and is important for selecting good candidate points for matching. Of the interest operators discussed in Chapter III, the Moravec Operator (IOP2) provides distinct points because of the way it selectively isolates the local maxima of the directed variances. Similarity is a measure of how closely two points resemble one another. Finally, consistency is a measure of how well a particular match (that is, a particular disparity classification) conforms to nearby matches. The continuity of disparity over most of the image can be used to avoid false matches based on similarity alone by suppressing matches in the absence of supporting local evidence [Ref. 4: 334]. The discreteness, similarity, and consistency properties all may interact to enhance the overall performance of the matching algorithm.

To perform matching, the first step is to find the interesting points in the two images to be matched. This is accomplished by applying the interest operator independently to each image, and rejecting points with locally maximal but very small variances by adjusting the threshold value. This threshold should be adjusted to produce a reasonable number of points on each image, expressed as a percentage of the total number of image pixels.

After the two sets of interesting points (now called candidate points) are found, the next step is to construct a set of possible matches. An initial set of possible matches is constructed by pairing each candidate point from image 1 with every candidate from image 2 within some maximum distance (disparity window) of the (x,y) location of the point in image 1. This distance is the maximum detectable disparity in the x or y direction. The set of possible matches is organized as a collection of "nodes" $\{a_i\}$. One node exists for each candidate point from image 1. Associated with each node a_i is a set of coordinates (x_i, y_i) which is the location of the point in image 1, and a set of labels L_i which represent possible disparities that may be assigned to the point. Each label in L_i is either a disparity vector (l_x, l_y) , or it is a distinguishing label l^* , denoting an "undefined disparity." A node a_i has an undefined disparity if point (x_i, y_i) in image 1 does not correspond to any candidate point in image 2. The point (x_i, y_i) in image 1 is tentatively matched to a point (x,y) in image 2 within the disparity window of a_i by entering a label $l = (x_i - x, y_i - y)$ into L_i . For every node a_i , an estimated probability $p_i(l)$ is associated with every label $l = (l_x, l_y)$ in L_i . This is the probability that the point (x_i, y_i) in image 1 has disparity l . Thus, $p_i(l)$ is in $[0,1]$, and the sum of the individual label probabilities must equal 1. As mentioned above, these probability estimates will be successively improved by applying the consistency property.

The initial probabilities $p_i^0(l)$ are based on an algorithm which compares the correlation windows of the candidate points from image 1

with those of image 2. When these correlation windows are similar, the difference between them will be small. In this case, $p_i^0(1)$ should be large. Thus, if $s_i(1)$ is a measure of the difference between the correlation windows, then a weight $w_i(1)$ may be defined as

$$w_i(1) = \frac{1}{1+c s_i(1)} \quad (1 \neq 1^*) \quad (35)$$

This is the weight associated with the label 1 of node a_i (c is merely an arbitrary constant). Note that if no label has a high weight, then there is probably no valid match. Therefore, the initial estimate of the probability that the point (x_i, y_i) in image 1 corresponds to no point in image 2 is given by

$$p_i^0(1^*) = 1 - [\text{MAX}_{1 \neq 1^*} w_i(1)] \quad (36)$$

Next, Bayes' Rule can be applied to determine the conditional probability that a_i has label 1 given that a_i is matchable. This conditional probability can be estimated as

$$p_i(1|i) = \frac{w_i(1)}{\sum_{1 \neq 1^*} w_i(1)} \quad (37)$$

The initial probability that a_i should be label 1 for labels other than 1^* is then

$$p_i^0(1) = p_i(1|i)[1-p_i^0(1^*)] \quad (1 \neq 1^*) \quad (38)$$

Equations (35), (36), (37), and (38) can be used to calculate the initial probabilities for every label of every node.

The initial probabilities, which depend only on the similarity of

the candidate matching points, can be improved by using the consistency property. To update the initial probabilities, the new probabilities $p_i^{k+1}(1)$ should tend to increase when nodes with highly probable labels consistent with 1 are found near a_i . Labels are considered if they represent nearly the same disparities as given by

$$||1 - 1'|| \leq T \quad (39)$$

for an appropriate threshold, T. A node a_j may be considered near a_i if

$$\text{MAX} \{ |x_i - x_j|, |y_i - y_j| \} \leq N \quad (40)$$

where N is the consistency neighborhood for node a_i .

The degree to which the label $1'$ of a_j reinforces $p_i(1)$ is directly related to the estimated likelihood that $1'$ is correct. To compute the updated probabilities, a consistency factor is defined as

$$q_i^k(1) = \sum_{\substack{a_j \text{ near } a_i \\ j \neq i}} \{ \sum_{||1-1'|| \leq T} p_j^k(1') \} (1 \neq 1*) \quad (41)$$

This factor is zero if no nodes surrounding a_i have possible matches with disparity labels similar to 1. In like manner, a consistency factor for $p_i(1*)$ is defined as

$$q_i^k(1*) = \sum_{\substack{a_j \text{ near } a_i \\ j \neq i}} p_j^k(1*) \quad (42)$$

Probability updates may now be accomplished using the old probabilities and the computed consistency factors as

$$\hat{p}_i^{k+1}(1) = p_i^k(1) [A + Bq_i^k(1)] (1 \neq 1*) \quad (43)$$

and, for l^*

$$\hat{p}_i^{k+1}(l^*) = p_i^k(l^*)[A + Bq_i^k(l^*)] \quad (44)$$

The parameters A and B are positive constants which influence the convergence characteristics of the model. The role of A is to delay the total suppression of unlikely labels. Even if $q_i(l)$ is zero, the value of A ensures that the new probability does not become zero immediately. The role of B is to determine the rate of convergence. The larger B is relative to A , the faster will be the convergence of the disparity assignments. Thus, A and B may be interpreted as damping and gain parameters respectively.

Finally, the $\hat{p}$'s are normalized to obtain the new probabilities by

$$p_i^{k+1}(l) = \frac{\hat{p}_i^{k+1}(l)}{\sum_{l' \in L_i} \hat{p}_i^{k+1}(l')} \quad (45)$$

and, for l^*

$$p_i^{k+1}(l^*) = \frac{\hat{p}_i^{k+1}(l^*)}{\sum_{l' \in L_i} \hat{p}_i^{k+1}(l')} \quad (46)$$

The probability update procedure is successively iterated until the network reaches a steady state condition; but, for the purposes of this effort, the update procedure was arbitrarily stopped at 10 iterations. Upon termination of the update process, the label probabilities are tested. Those nodes with a disparity label having an estimated likelihood 0.7 or greater are considered to be matched. The disparity label with this high probability is then added to the coordinates of its associated node to determine the coordinates of the

point in image 2 to which it is matched. Note, however, that some nodes may remain ambiguous. That is, they may retain several potential matches having nonzero probabilities, all of which will be less than 0.5. In this case, it is impossible to determine the proper match[Ref. 4:333-336].

In program DIDA, to match interesting points using the relaxation-labeling matching algorithm, the user enters a 1 when queried concerning the desired matching algorithm in subroutine MTASEL, as explained in the interesting point matching section of Chapter II above. This entry causes MTASEL to call subroutine MTALG1 which matches the interesting point sets being compared.

To begin with, subroutine MTALG1 calls subroutine DWDSZE to allow the user to specify the size of the disparity window to be used to form the disparity labels for each node. Specification of the disparity window size is accomplished in exactly the same manner as the specification of the processing window and correlation window sizes. Subroutine DWDSZE calls subroutine DRCFND to obtain the number of disparity window rows and columns. DRCFND then calls subroutines DWROW and DWCOL to permit the user to interactively enter the number of disparity window rows and columns respectively. Subroutines DRCCHK and DRCCOR are used for entry error checking and correction, if needed. The flow diagram for inputting the disparity window row and column size is identical to the one shown in Figure 6.

The information obtained after calling DWDSZE is stored in the

area COMMON /DSPWND/. The variables DWDROW, DWDCOL, DWDRWH, DWDCLH, DWDSZE, and DWDINV specify exactly the same information as the other windows mentioned above. The range of values for the disparity window to be specified is shown in Table I.

After the disparity window size has been specified, subroutine MTALG1 next calls subroutine CWCOMP to determine the orientation of the correlation windows of the two interesting point sets to be matched. This is accomplished by comparing the correlation window row and column sizes of each window to see if they are less than, equal to, or greater than each other. Thus, since there are three possible conditions for each case, and two tests per case (rows and columns), there are $3^2 = 9$ possible orientations. CWCOMP assigns a value to the variable CWTEST based on the orientation of the two correlation windows (values range from 0 to 8), and returns this value to the calling program.

Next, MTALG1 calls subroutine W8ASEL to allow the user to select the weighting algorithm to be used to form the initial probabilities for matching. Subroutine W8ASEL operates in the same fashion as the other previously described selection subroutines (such as MTASEL). However, instead of then calling the selected subroutine, W8ASEL merely returns a value in the variable W8ANBR corresponding to the weighting algorithm selected. This number is passed to subroutine LABELS, and from LABELS to subroutine W8AFND to actually call the weighting algorithm the user has selected. This technique of assigning the

algorithm number in one subroutine and passing it to another subroutine which calls for that algorithm is called "transference." After the weighting algorithm number has been assigned, subroutine CONST is called which permits the user to interactively select the constant, C, to be used in the weighting algorithms (except for algorithm 6, as explained later). As usual, error checking and correction is provided in CONST.

When the disparity window size has been established, the orientation of the correlation windows determined, and the weighting algorithm (and constant) selected, subroutine MTALG1 calls subroutine LABELS to begin the matching process. LABELS determines the disparity labels corresponding to each node formed by interesting point set 1 by differencing the x and y coordinate values of each node with the interesting points from set 2 lying within the disparity window formed by that node. After a label is determined, subroutine CWDEQ8 is called to equate the correlation window sizes surrounding the interesting points from each set according to the orientation determined in subroutine CWCOMP (given by the variable CWTEST). Once these correlation window sizes are equated, they may be directly compared. Subroutine LABELS next calls the weighting algorithm specified by the value of W8ANBR to compute a weight for determining the initial probability to be assigned to each label. LABELS then returns to MTALG1.

Next, MTALG1 calls subroutine INPROB to determine the initial

probabilities of each label according to the weight computed in subroutine LABELS. After this, subroutine NGHSZE is called to determine the consistency neighborhood to be used to test the consistency property of each node with other nodes which are close-by. Subroutine NGHSZE calls subroutine NRCFND to obtain the number of consistency neighborhood (hereafter referred to as just neighborhood) rows and columns. NRCFND then calls subroutines NGROW and NGCOL to permit the user to interactively enter the number of neighborhood rows and columns respectively. Subroutines NRCCHK and NRCCOR are used for entry error checking and correction, if needed. The flow diagram for inputting the neighborhood row and column size is identical to the one shown in Figure 6.

The information obtained after calling NGHSZE is stored in the area COMMON /NBHOOD/. The variables NGHROW, NGHCOL, NGHRWH, NGHCLH, NGHSZE, and NGHINV specify exactly the same information as the other windows mentioned above. The range of values for the neighborhood to be specified is shown in Table I.

After specifying the neighborhood, MTALGI calls subroutine CONSTS to allow the user to interactively input the constants A and B used to compute the consistency factor for the probability updates. Subroutine THRESH is then called to allow the user to interactively input the threshold, T, for the label consistency tests. As usual, error checking and correction is provided, if needed.

Finally, subroutine MTALGI calls subroutine UDPROB to update the

initial probabilities based on the consistency property. Subroutine UDPROB tests the labels of the nodes within the neighborhood for consistency by comparing the absolute value of the labels of each node pair with the user-defined threshold. For label differences smaller than this threshold, the consistency factor associated with that label probability is modified accordingly. After all the labels of all the nodes have been tested, the label probabilities for each node are updated, and the no-match probability for each node is updated. When the probability updates reach steady state (10 iterations are used for this study), UDPROB returns to MTALG1. This completes the matching process and subroutine MTALG1 passes control to subroutine IPMTCH.

Label and Weight Determination

To determine the labels and weights associated with each node, subroutine MTALG1 calls subroutine LABELS, as mentioned above. LABELS uses the value of N (see Table I) to determine the storage locations and maximum allowable number of the labels and weights. To store these labels and weights, the VIRTUAL array STORE(N,M) is used. Note that since the weights (and later, the probabilities) are in the range [0,1], STORE must be a REAL*4 array; thus, its size is limited to N x M (32 x 512). Each column of array STORE contains the information for each node formed from the interesting points of interesting point set 1. Thus, M nodes can be stored.

The first two row positions of array STORE are used to store the row and column coordinates (x_i, y_i) of each node $\{a_i\}$. The third row is

used to store the label count (total number of labels and weights stored). The labels are stored beginning in the fourth row of STORE. These labels are stored in pairs with the x-coordinate difference first, followed by the y-coordinate difference. These differences are given by the variables LX and LY respectively. The x_i and y_i coordinate values of each node are given by the variables IPROW1 and IPCOL1 respectively. The weights are stored beginning in the row immediately following the storage location for LY for the last label. Since two storage locations are required for each label, only half as many storage locations are required for the weights as for the labels. The weights are given by the variable W. The last row of array STORE is reserved for the no-match probability $p_i(1^*)$.

From the above information, the starting row location for storing the weights in array STORE can be determined as a function of N. Since the first three and last rows are used for other information, $N-4$ rows are available for label and weight storage. There are two thirds as many labels as weights. Thus, since the first three storage locations are used, the last label storage location is row $2(N-4)/3+3$. Therefore, the row starting location of the first weight is $2(N-4)/3+4$. Therefore, $2(N-4)/3$ labels and $(N-4)/3$ weights will be stored in array STORE. These values are, of course, truncated to the corresponding integer value. Thus, one or more storage locations may not be used, depending on the value of N. The variable LBLTST is assigned the value of the last label row storage position to test for an excessive number of labels during program execution. For $N=32$, there is space for 9

labels (18 storage locations) and 9 weights. Note that adding the 4 additional locations required for storing the other variables yields 31 locations with one location unused (this will be row 31). The relative locations of the variables stored in array STORE is shown in Table V.

Subroutine LABELS begins by reading the row and column values for the first node from IP set 1 file, and stores the correlation window in array IPCW1. Next, the number of interesting points in IP set 2 is read from IP set 2 file, and the values of the x and y coordinate locations of node a_1 (IPROW1 and IPCOL1) are placed in array locations STORE(1,1) and STORE(2,1) respectively. The disparity window is formed around node a_1 by adding and subtracting the values of DWDRWH and DWDC LH to IPROW1 and IPCOL1 to obtain the window boundaries DRWMIN, DRWMAX, DCLMIN, and DCLMAX as shown below:

DRWMIN					
IPROW1			a_1		
DRWMAX					
	DCLMIN	IPCOL1	DCLMAX		

Interesting points from IP set 2 are now read in successively, along with their associated correlation windows. The x and y coordinate values of these points are given by IPROW2 and IPCOL2 respectively. The correlation window is stored in array IPCW2. The interesting point row and column locations of IP set 2 are tested

Table V. Relative Locations of Labels and Weights in Array STORE

ARRAY STORE(N,M)				
Variables Associated with node i	Nodes of IP SET 1			
	1	2	...	M
1. Row (X_i) of IP(i)	IPROW1			
2. Column (Y_i) of IP(i)	IPCOL1			
3. Number of Labels (j)	LBLKNT			
4. <u>LABELS</u> l_{x1}	LX			
5. l_{y1}	LY			
6. l_{x2}				
7. l_{y2}				
• •				
• •				
• •				
• •				
• •				
• •				
2(N-4)/3+2. l_{xj}				
2(N-4)/3+3 l_{yj}				
2(N-4)/3+4. $w_1(1)$	W			
2(N-4)/3+5. $w_2(1)$				
• <u>WEIGHTS</u> •				
• •				
• •				
• •				
• •				
• •				
• •				
• •				
2(N-4)/3+(j+3). $w_j(1)$				
2(N-4)/3+(j+4). •••••	(UNUSED IF 2(N-4) NOT DIVISIBLE BY 3)			
N. $P_i(1^*)$				

against the values of DRWMIN, DRWMAX, DCLMIN, and DCLMAX to determine if any of the points lie within the disparity window. If both IPROW2 and IPCOL2 lie within the bounds of these variables, that interesting point becomes a candidate match point, and the flags DRWFLG and DCLFLG are set to 1 to signal the program to form a label and associated weight using this point. To determine the weight, subroutine CWDEQ8 is first called to equate the correlation window sizes, and then subroutine W8AFND is called to obtain the weighting algorithm specified earlier in subroutine W8ASEL. This weighting algorithm determines the weight associated with the label being formed.

After the weight has been determined, the label associated with the candidate match point is formed by

$$LX = IPROW1 - IPROW2 \quad (35)$$

$$LY = IPCOL1 - IPCOL2 \quad (36)$$

This label is then stored in the appropriate row position in STORE, W is stored in its appropriate row position, and LBLKNT is incremented by 1. When every node from IP set 1 has been compared with every interesting point from IP set 2, and the labels and weights formed from each candidate match point that lies within the disparity window, then the complete set of labels and weights for all the nodes has been specified.

If there are more candidate match points within the disparity window of a given node than label storage positions, the label storage size of array STORE will be exceeded. This condition is detected by the variable LBLTST. When this condition occurs, an advisory message

is displayed to the user, along with the row and column locations of both interesting point sets indicating where the condition occurred. The disparity window is then reduced in size by one pixel in each direction by adding 1 to DRWMIN and DCLMIN and subtracting 1 from DRWMAX and DCLMAX. The same set of interesting points is then again compared with the same node using this smaller window size. If the label storage size is again exceeded, the disparity window is reduced in size again, and the same comparison sequence is repeated. This process continues until the disparity window is made small enough to prevent the number of labels from becoming excessive, or until the disparity window reaches its minimum size (which is 2x2). In either case, processing then proceeds to the next node. This technique of successively reducing the window size is called "adaptive windowing."

Equating Correlation Window Sizes. In order to adequately compare the correlation windows of the nodes from interesting point set 1 and the candidate match points from interesting point set 2 to determine a weight, the intersection of these two windows (that is, the points which both windows have in common) must be determined. To make this determination, subroutine LABELS calls subroutine CWDEQ8 which equates the sizes of the two correlation windows from each interesting point based on the value of the orientation variable CWTEST. Eight different cases are possible (the case where the rows and columns of both windows are equal is not needed since the window sizes are already equated). For example, consider the case where the correlation windows of the interesting points of IP set 1 are 8 X 9 (8 rows and 9 columns), and

the correlation windows of the interesting points of IP set 2 are 14 X 4 as shown in Figure 26. This is the case where CWROW1 is less than CWROW2 and CWCOL1 is greater than CWCOL2. This case would be specified in subroutine CWDEQ8 by a value of CWTEST = 4. Note that in this case, both correlation windows overlap each other.

For the case mentioned above, CWDEQ8 first forms the variables

IRWDIF = CWROW2 - CWROW1	(37)
IRWDFH = IRWDIF/2	(38)
ICLDIF = CWCOL1 - CWCOL2	(39)
ICLDFH = ICLDIF/2	(40)

Using these variables, the elements which overlap each other in the two correlation windows (which are aligned centered on each interesting point) are stored in the temporary arrays ITMP1 and ITMP2. These elements will be the ones shown in the shaded area of Figure 26. The other elements will be discarded. After all the elements from both windows have been stored in the temporary arrays, they are restored in their original arrays IPCW1 and IPCW2, and are now of identical size. Since the correlation windows are now of identical size, and centered about their respective interesting points, they may be directly compared by the weighting algorithms. The other 7 cases are treated in the same manner as this one. If the debug version of program DIDA is being executed, these new correlation windows are printed on the line printer after they are formed.

Weighting Algorithms. Once the correlation window sizes have been equated by subroutine CWDEQ8 (if necessary), subroutine LABELS calls

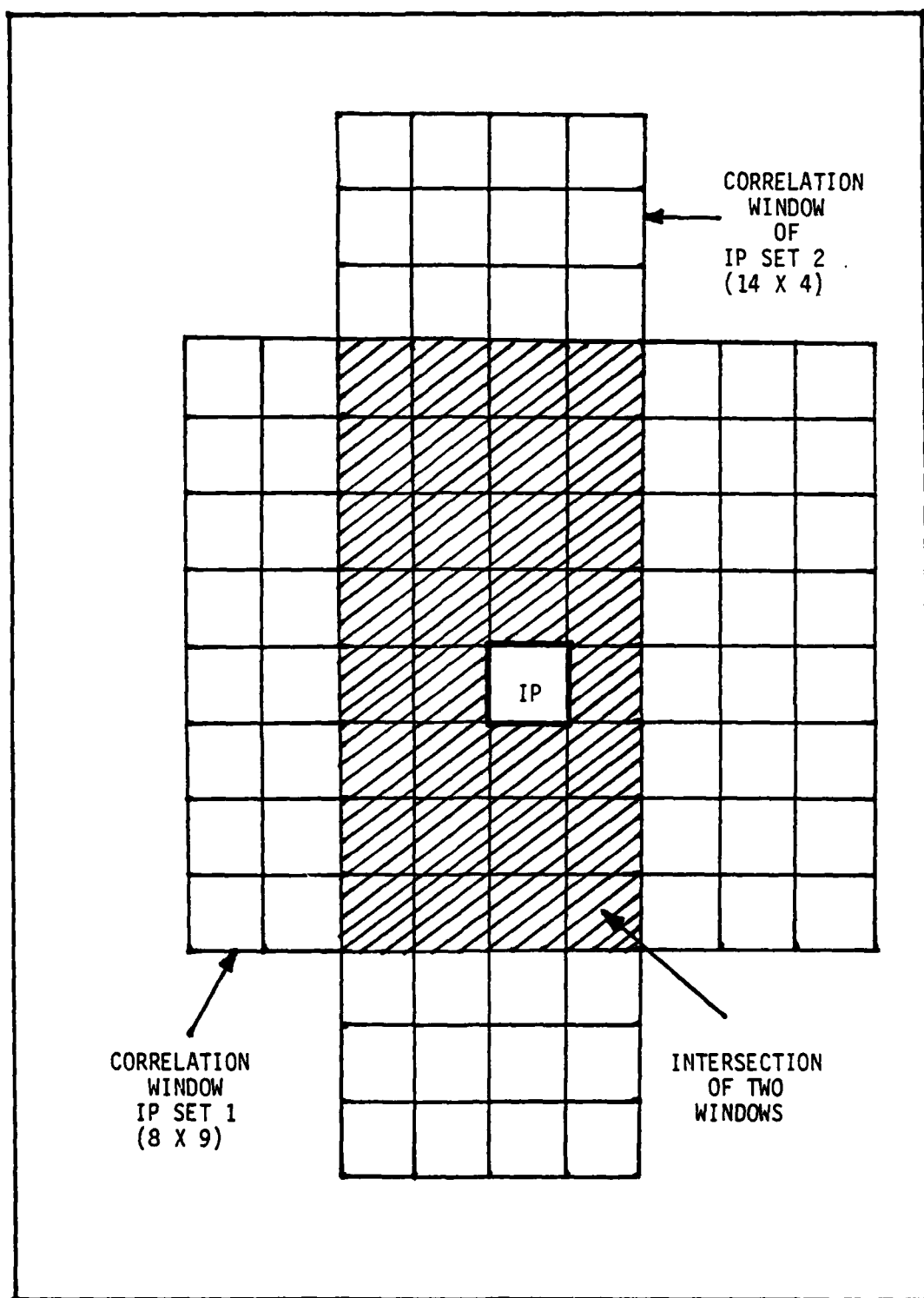


Figure 26. Orientation of an 8X9 and 14X4 Set of Correlation Windows by Subroutine CWDEQ8.

subroutine W8AFND to in turn call the subroutine associated with the weighting algorithm specified by the user in subroutine W8ASEL. This weighting algorithm subroutine is called based on the value of W8ANBR transferred by subroutine W8ASEL. Note here the formidable power of the transference principle. Even though the user specified this algorithm much earlier in the program, it is as though it has just been specified as far as the program is concerned.

Six weighting algorithms may be selected for use in program DIDA: (1) sum-of-squares pixel differencing; (2) average difference; (3) center-minus-average difference; (4) gradient difference; (5) variance difference; and (6) statistical correlation. Each of these weighting algorithms is described below.

Sum-of-Squares Pixel Differencing. In this weighting algorithm (W8ALG1), the difference of the pixels of the correlation windows of the two interesting points being compared is obtained. This difference is given by the variable CWNDF. The square of this difference is then obtained as given by the variable DIFSQR. The sum of these squared differences is then obtained over the window area as given by the variable SUM. The value of the weight, W, returned to LABELS is then given by equation (35) as

$$W = 1.0 / (1.0 + C * SUM) \quad (41)$$

where C is the arbitrary constant previously defined. This is the method of weighting used by Barnard and Thompson [Ref. 4: 335].

Average Difference. In this weighting algorithm (W8ALG2), the sum of the elements of each correlation window is obtained. This sum is then used to obtain the average by dividing by the window size. The sums are given by the variables SUM1 and SUM2 and the corresponding averages are CWAVE1 and CWAVE2. The absolute value of the difference of these two averages is then used to form the weight as given by equation (41) above.

Center-Minus-Average Difference. In this weighting algorithm (W8ALG3), the sum of each correlation window is formed as in W8ALG2, and the average is obtained. The value of the average of each window is then subtracted from the center element of the window. The absolute value of the difference of this result for each window is used to form the weight as given by equation (41).

Gradient Difference. In this weighting algorithm (W8ALG4), the gradient of each correlation window is obtained by first forming a square window from the existing window. Four sums are then formed: (1) upper left quarter of window; (2) upper right quarter of window; (3) lower left quarter of window; and (4) lower right quarter of window. These sums are specified for both windows by the variables ASUM1, ASUM2, BSUM1, BSUM2, CSUM1, CSUM2, and DSUM1, DSUM2. The variable ISQRT describes the new window size.

Next, four coordinate averages are formed by dividing the quadrant sums by the number of elements of that quadrant. The displacement between quadrants is given by

$$NDF = (ISQRT + 1)/2 \quad (42)$$

and the coordinate differences for each window are given by

$$ALPHA1 = ((A1 + B1) - (C1 + D1))/NDF \quad (43)$$

$$BETA1 = ((B1 + D1) - (A1 + C1))/NDF \quad (44)$$

$$ALPHA2 = ((A2 + B2) - (C2 + D2))/NDF \quad (45)$$

$$BETA2 = ((B2 + D2) - (A2 + C2))/NDF \quad (46)$$

where A1, B1, C1, and D1 are the quadrant averages for the first window, and A2, B2, C2, and D2 are the quadrant averages for the second window. The gradients of each window are then given by

$$G1 = SQRT(ALPHA1 * ALPHA1 + BETA1 * BETA1) \quad (47)$$

$$G2 = SQRT(ALPHA2 * ALPHA2 + BETA2 * BETA2) \quad (48)$$

The absolute value of the difference of these two gradients is then used to form the weight as given by equation (41).

Variance Difference. In this weighting algorithm (W8ALG5), the variance of each correlation window is computed by first forming the sum and sum-of-squares of the elements of each window. These sums are given by the variables SUM1, SUM2, and SQRSM1, SQRSM2. The means are then formed and given by the variables CMEAN1 and CMEAN2. The mean squares are given by SQRCM1 and SQRCM2. Finally, the variances of the windows are given by

$$CWVAR1 = SQRSM1/CWSZE - SQRCM1 \quad (49)$$

$$CWVAR2 = SQRSM2/CWSZE - SQRCM2 \quad (50)$$

The absolute value of the difference of these variances (given by the variable CVRDIF) is then used to form the weight as given by equation

(41).

Statistical Correlation. In this weighting algorithm (W8ALG6), the statistical correlation is determined for the two correlation windows by the expression [Ref. 21: 91].

$$\hat{r}_{xy} = \frac{\hat{C}_{xy}}{(V_x V_y)^{1/2}} = \frac{N \sum_{i=1}^n x_i y_i - (\sum_{i=1}^n x_i)(\sum_{i=1}^n y_i)}{([N \sum_{i=1}^n x_i^2 - (\sum_{i=1}^n x_i)^2][N \sum_{i=1}^n y_i^2 - (\sum_{i=1}^n y_i)^2])^{1/2}} \quad (51)$$

where $-1 \leq \hat{r}_{xy} \leq 1$ is the statistical correlation coefficient. Here, to form the desired weight, W , we merely set $W = \text{ABS}(\hat{r}_{xy})$. To compute the statistical correlation coefficient, the same sums and sums-of-squares are formed as in W8ALG5, except this time the cross terms must also be included. These cross-product terms are given by the variables CRSPRD and CPSUM. The sum squares for each window are given by the variables SQRCS1 and SQRCS2. The covariance contribution (numerator of equation (51)) is then

$$C12 = \text{CWSIZE} * \text{CPSUM} - \text{SUM1} * \text{SUM2} \quad (52)$$

and the individual variances are given by

$$V1 = \text{SQRSM1} * \text{CWSIZE} - \text{SQRCS1} \quad (53)$$

$$V2 = \text{SQRSM2} * \text{CWSIZE} - \text{SQRCS2} \quad (54)$$

Finally, the weight is given by

$$W = \text{ABS}(C12 / \text{SQRT}(V1 * V2)) \quad (55)$$

Initial Probabilities

After determination of the disparity labels and weights in subroutine LABELS, subroutine MTALG1 calls subroutine INPROB to determine the initial label probabilities from the previously determined weights. To begin with, INPROB determines the sum of the weights and maximum weight for node a_i , as given by the variables WSUM and WMAX respectively. The no-match probability is then given by equation (36) as (array STORE is renamed P in this subroutine)

$$P(N,i) = 1 - WMAX \quad (56)$$

The conditional probability is given by equation (37) as

$$CPROB = P(1,i) / WSUM \quad (57)$$

for each label, i . Finally, the initial probabilities for each label are determined from equations (56) and (57) as given by equation (38) as

$$P(1,i) = CPROB * (1.0 - P(N,i)) \quad (58)$$

If the debug version of program DIDA is being executed, these initial probabilities, and the no-match probability, are printed on the line printer.

Probability Updates

Once the initial probabilities have been determined, they are updated by MTALG1 by calling subroutines UDPROB. Subroutine UDPROB tests for nodes within the consistency neighborhood defined by the

variables NRWMIN, NRWMAX, NCLMIN, and NCLMAX in exactly the same fashion as subroutine LABELS tested for the interesting points from IP set 2 within the disparity window. To begin with, the consistency factors QL(1) and QLSTR are set to zero. Then, for node a_i the variables $NROW1 = P(1,i)$ and $NCOL1 = P(2,i)$ are defined to form the consistency neighborhood (again, array STORE is renamed P in this subroutine). Thus, adding and subtracting the values of NGHRWH and NGHCLH to NROW1 and NCOL1 forms the neighborhood. As in the case of the disparity window, nodes a_j are tested for presence within the neighborhood formed by a_i (note: $j \neq i$).

Next, UDPROB forms the variables $NROW2 = P(1,j)$ and $NCOL2 = P(2,j)$ to test for the presence of node a_j within the consistency neighborhood. If a node is found within this neighborhood, the flags NRWFLG and NCLFLG are set to 1 to signal the presence of this node in the neighborhood. In this case, the no-match consistency property QLSTR is updated in accordance with equation (42) as

$$QLSTR = QLSTR + P(N,j) \quad (59)$$

After this, the label differences between nodes a_i and a_j are compared in accordance with equation (39) as

$$LBLRDF = \text{ABS}(P(1_x,i) - P(1_x,j)) \quad (60)$$

$$LBLCDF = \text{ABS}(P(1_y,i) - P(1_y,j)) \quad (61)$$

Each of these label differences is then tested against the user-defined threshold, T. If LBLRDF is less than or equal to T, the flag LBRFLG is

set to 1 to signal that this difference is less than or equal to the threshold value. Similarly, if LBLCDF is less than or equal to T, LBCFLG is set to 1. If both flags are simultaneously set to 1 by a given label test, the consistency factor is updated in accordance with equation (41) as

$$QL(1) = QL(1) + P(1,j) \quad (62)$$

The updated probabilities are next formed in accordance with equations (43) and (44) as

$$P(1,i) = P(1,i) * (A + B * QL(1)) \quad (63)$$

and for l^*

$$P(N,i) = P(N,i) * (A + B * QLSTR) \quad (64)$$

The probability sum is also formed while these probabilities are being updated as

$$PSUM = PSUM + P(1,i) \quad (65)$$

and, for l^*

$$PSUM = PSUM + P(N,i) \quad (66)$$

Finally, the new probabilities (which are really the updated probabilities) are formed in accordance with equations (45) and (46) as

$$P(1,i) = P(1,i) / PSUM \quad (67)$$

and for l^*

$$P(N,i) = P(N,i)/PSUM \quad (68)$$

If the debug version of program DIDA is being executed, these new (updated) label probabilities, along with the new no-match probabilities, are listed on the line printer. The iteration number is also printed at the beginning of the probability update.

Thus, in summary, to match interesting points selected on one image to those selected on a second image using the relaxation-labeling matching algorithm, a disparity window is first formed by the nodes of image 1. Interesting points from image 2 that lie within this disparity window are used to form labels by differencing their x-y coordinate values with those of the node that formed the window. Weights are formed using the correlation windows of the interesting points from image 2 within the disparity window and the correlation window of node a_1 . These weights are formed by an algorithm which the user selects. Next, the initial probabilities, and initial no-match probability, are formed using the previously determined weights. These probabilities are updated based on the consistency property by comparing nodes within a given neighborhood of one another. If these nodes have similar labels, the consistency factor is increased. Also, the no-match consistency factor is updated for nodes which are close-by. Thus, if nodes within the consistency neighborhood have no labels that are similar, the no-match consistency factor will increase, but the match consistency factor will not. However, if several labels

within the neighborhood are similar, the match consistency factor will increase more than the no-match consistency factor. Finally, the probability updates are accomplished using these consistency factors, and the new probabilities are obtained by dividing these probability updates by the probability sum (including 1*). A flow diagram illustrating the relaxation-labeling matching algorithm is shown in figure 27.

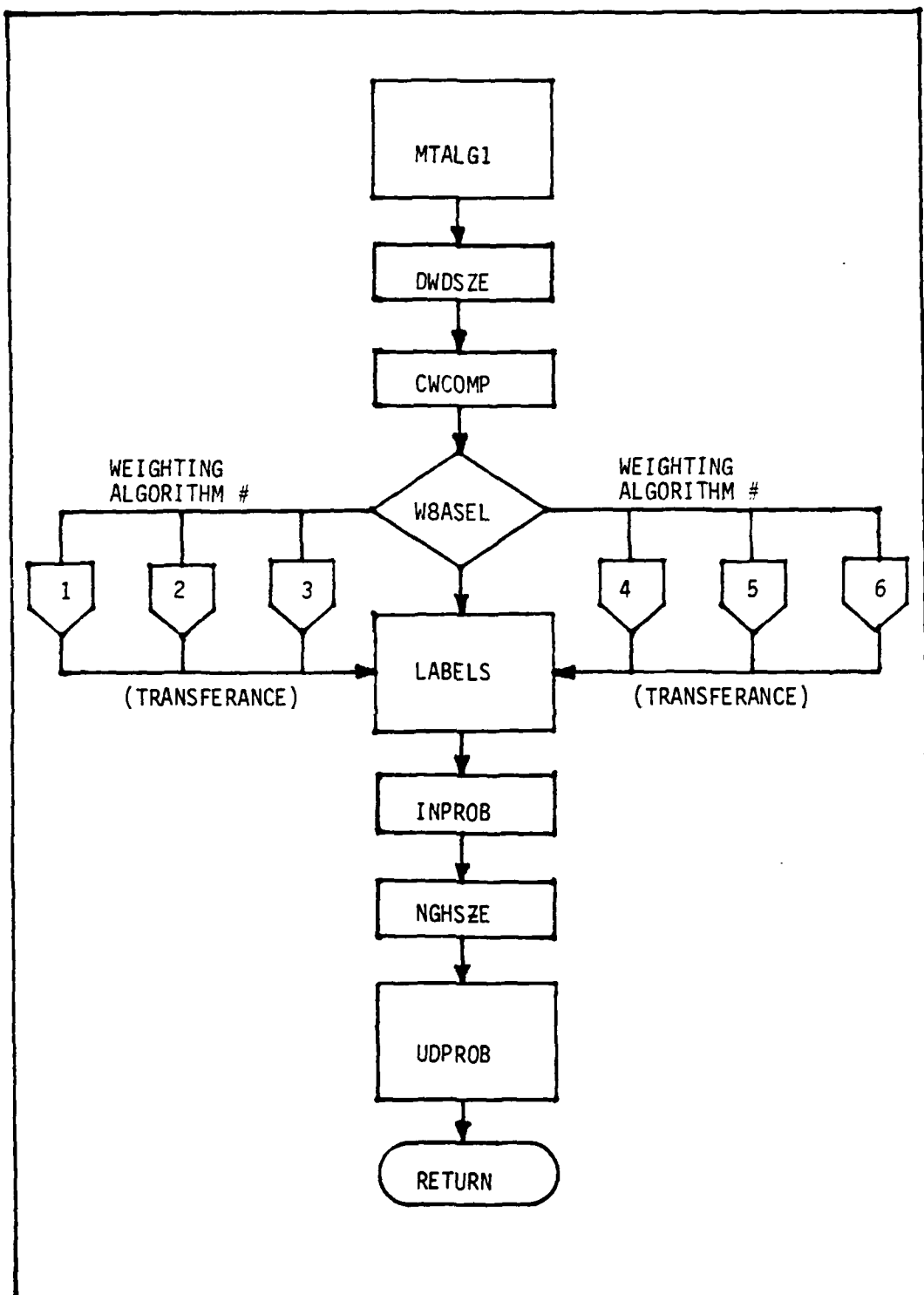


Figure 27. Relaxation-Labeling Matching Algorithm Flow Diagram

V. Results and Conclusions

During the course of this study, an interactive program, called program DIDA, was developed to be used to perform disparity analysis of time-varying imagery. The program consists of four major sections: (1) image operations; (2) interesting point selection; (3) interesting point matching; and (4) image display. Three data bases were obtained and structured to produce the proper format for use by program DIDA. These data bases are: (1) the NVL data base; (2) the UM data base; and (3) the UH data base. Photographs of selected images from each data base were obtained. Also, photographs of selected images with interesting points overwritten on them were obtained.

Each of the four sections of program DIDA were tested to determine the effectiveness of their operation. Although time did not permit an exhaustive analysis of each section, representative operations from each section were performed to demonstrate the most useful features of each section. The results obtained from the operation of each section of program DIDA are given below, along with some concluding remarks.

Image Operations Results

Each of the image operations permitted by the image operations section of program DIDA was successfully tested. The operation to print image intensities on the line printer is particularly useful for checking the implementation of new data bases, and determining the threshold value to be used for interesting point selection. By

printing out a few rows of intensity values of a newly implemented data base, the correctness of the implementation can be confirmed. Also, by examining the intensity values of selected portions of the image, the variances can be approximated to determine an appropriate threshold to use to restrict the number of interesting points selected to a reasonable number.

The image creation and deletion operations can be used to test new interest operators and matching algorithms for correct operation before attempting to use them on large images. Using one of the image creation algorithms, a small test image can be created. Then, another small image can be created which is similar to the first one. The new interest operator is applied independently to both images to select interesting points. The new matching algorithm is used to attempt to match these selected points. During these operations, the debug version of the program is exercised to print out the intermediate processing steps during program execution. An analysis of the debug listing may then be used to determine errors in the interest operator algorithm, and the matching algorithm.

The debug version of program DIDA is exactly identical to the non-debug version. This version is called DIDAD, and prints out many intermediate computations performed during interesting point selection and matching that are not printed during the execution of the non-debug program version. This debug output is produced by placing a "D" in column one of the FORTRAN code line. At compile time, when the

compiler "sees" this D in the first column, it is ignored unless the /DE switch has been specified. If this switch is specified, the compiler does not ignore the statements with a "D" in column one, and these statements are executed when the program is run. Thus, two program versions were compiled; one with the /DE switch specified (DIDAD) to produce the debug version, and another without this switch specified (DIDA) to produce a non-debug version.

After the new interest operator has been tested using the debug program version on a small, test image, or a new matching algorithm has been tested on two or more small test images, any errors which appear can be corrected using the debug listing. The non-debug version of the program may then be executed on much larger images to obtain meaningful results. When they are no longer needed, the small test images can be deleted using the image deletion operation.

Interest Operator Results

Each of the interest operators implemented in program DIDA were tested for correct operation using the debug version of the program, as mentioned above. These operators were then applied to two of the images from the UM data base for various threshold values. It was found that the number of interesting points selected is inversely proportional to the threshold value specified. This is to be expected since there will be, in general, many more points with low variance values than points with high values. Thus, for high threshold values, only a few variance values should exceed this threshold and become

interesting points. Conversely, for low threshold values, many interesting points should be selected.

Another result observed was that the threshold value required to produce the same number of interesting points varied for each of the three different interest operators. In general, a lower threshold value was required for the directed variance and edged variance interest operators than for the simple variance operator to produce the same number of interesting points. The effect of changing the threshold value on the number of interesting points produced for two different image pairs from the UM data base is shown in Table VI. The interest operators included in this table, according to their number, are:

IOP#1 = Simple Variance Operator
IOP#2 = Directed Variance Operator
IOP#3 = Edged Variance Operator

Matching Algorithm Results

The relaxation-labeling matching algorithm implemented in program DIDA (MTALG1) was tested for correct operation using the debug version of the program, as mentioned above. This algorithm was difficult to adequately test due to a lack of sufficiently similar test images. When tested on larger images using the non-debug program version, very poor matching efficiencies were observed. The constants for the matching algorithm suggested by Thompson [Ref. 4:336] were used throughout the test. In every test conducted, very low matching

Table VI. Effect of Changing the Variance Threshold on the Number of Interesting Points Produced.

#	From Image	IPS Name	IOP #	Process Window Size	Correlation Window Size	Threshold	ϵ	# IPS
1.	TRUCK.IMG;1	TRUCK.IPS;1	2	11 X 11	11 X 11	500	1	266
2.	TRUCK.IMG;2	TRUCK.IPS;2	2	11 X 11	11 X 11	500	1	284
3.	TRUCK.IMG;1	TRUCK.IPS;3	1	11 X 11	11 X 11	5000	-	427
4.	TRUCK.IMG;2	TRUCK.IPS;4	1	11 X 11	11 X 11	5000	-	465
5.	TRUCK.IMG;1	TRUCK.IPS;5	3	11 X 11	11 X 11	5000	-	7
6.	TRUCK.IMG;2	TRUCK.IPS;6	3	11 X 11	11 X 11	4000	-	27
7.	TRUCK.IMG;1	TRUCK.IPS;7	2	11 X 11	11 X 11	2000	-	28
8.	TRUCK.IMG;2	TRUCK.IPS;10	2	11 X 11	11 X 11	2000	1	40
9.	TRAFIC.IMG;1	TRAFIC.IPS;1	2	11 X 11	9 X 9	2500	1	0
10.	TRUCK.IMG;1	TRUCK.IPS;11	2	11 X 11	9 X 9	1000	1	125
11.	TRUCK.IMG;2	TRUCK.IPS;12	2	11 X 11	9 X 9	1000	1	155
12.	PERSON.IMG;1	PERSON.IPS;1	2	11 X 11	9 X 9	1000	1	0
13.	PERSON.IMG;1	PERSON.IPS;2	2	11 X 11	9 X 9	10	1	287
14.	PERSON.IMG;1	PERSON.IPS;3	2	11 X 11	9 X 9	500	1	59
15.	PERSON.IMG;2	PERSON.IPS;4	2	11 X 11	9 X 9	500	1	54

efficiencies were observed (most were zero). The matching efficiencies of various interesting point sets obtained using the three interest operators of program DIDA are shown in Table VII (also refer to Table VI). It is believed that a more judicious selection of the parameters associated with the matching algorithm would produce much better results.

Image Display Results

Each of the image display options permitted by the image display section of the program DIDA was successfully tested. The blow-up feature worked well, but produced images that were somewhat distorted. The main source of distortion resulted from a "stairstep" effect along diagonal edges within the scene. This stairstep phenomena resulted from the fact that when the image is blown-up, each pixel is reproduced in an area surrounding itself the appropriate number of times to produce a 512x512 image (16 times for a 128x128 image).

Also, the colored images produced were not natural color. No pre-written algorithm was available with the RAMTEK driver to produce natural color images from a set of intensity values. Thus, the colors assigned to ranges of intensities were selected on a trial-and-error basis. The resulting colored images were more of a "pseudo-color" than naturally colored.

To illustrate the results of the interest operators, a number of photographs were taken of the two images from the UM data base

Table VII. Matching Efficiencies Obtained from Various Interesting Point Sets.

#	IP SETS		MATCHED	# POINTS IN SET			MATCHING CONSTANTS			# POINTS MATCHED	WEIGHTING ALGORITHM	MATCHING % EFFICIENCY
	1	2		1	2		A	B	C			
1.	TRUCK.IPS;7	TRUCK.IPS;10		28	40		0.3	3	-	1	6	3.571
2.	TRUCK.IPS;10	TRUCK.IPS;7		40	28		0.3	3	-	0	6	0
3.	PERSON.IPS;3	PERSON.IPS;4		59	54		0.3	3	10	0	1	0
4.	PERSON.IPS;3	PERSON.IPS;4		59	54		0.3	3	10	0	2	0
5.	PERSON.IPS;3	PERSON.IPS;4		59	54		0.3	3	10	0	3	0
6.	PERSON.IPS;3	PERSON.IPS;4		59	54		0.3	3	10	0	4	0
7.	PERSON.IPS;3	PERSON.IPS;4		59	54		0.3	3	10	0	5	0
8.	PERSON.IPS;3	PERSON.IPS;4		59	54		0.3	3	10	4	6	6.779661
9.	TRUCK.IPS;11	TRUCK.IPS;12		125	155		0.3	3	-	0	6	0

mentioned above. To begin with, a photograph of one of the images was taken in its normal size (128x128). The interesting points obtained from interest operator 2 were then written over the image, and it was photographed again. Next, the same image was blown up to 512x512 and photographed. Again, the interesting points were overwritten and the image was photographed. This sequence of photographs was obtained to illustrate how much easier it is to examine the interest operator results on large image as opposed to a much smaller one. These photographs are shown in Figures 28, 29, 30, and 31 respectively. Figures 32 and 33 show photographs of the same blown-up image with the results of interest operators 1 and 3 respectively overwritten.

A similar sequence was photographed for a different image as shown in Figures 34, 35, 36, and 37. Figure 38 shows the same image for a smaller number of interesting points (same interest operator using a higher threshold). Finally, photographs of one of the images from the UH data base and the NVL data base are shown in Figures 39 and 40 respectively. A summary of these photographs is given in Table VIII.

Conclusions

Although the results from the interest operators of program DIDA demonstrate an operational capability for interesting point selection, much work is still necessary before a full-scale real-time disparity analysis program can be realized. To begin with, the speed at which the interest operators execute must be greatly reduced. Typical processing times varied from approximately 15 minutes for a 128 x128

Table VIII. Summary of Imagery Photographs

FIGURE	IOP #	IMAGE NAME	IP SET NAME	NUMBER OF IP'S	SCALED
28	-	TRUCK.IMG;2	-	-	
29	2	TRUCK.IMG;2	TRUCK.IPS;2	284	NO
30	-	TRUCK.IMG;2	-	-	YES
31	2	TRUCK.IMG;2	TRUCK.IPS;2	284	YES
32	1	TRUCK.IMG;2	TRUCK.IPS;4	465	YES
33	3	TRUCK.IMG;2	TRUCK.IPS;6	27	YES
34	-	PERSON.IMG;1	-	-	NO
35	2	PERSON.IMG;1	PERSON.IPS;2	287	NO
36	-	PERSON.IMG;1	-	-	YES
37	2	PERSON.IMG;1	PERSON.IPS;2	287	YES
38	2	PERSON.IMG;1	PERSON.IPS;3	59	YES
39	-	TRAFIC.IMG;1	-	-	NO
40	-	TERAIN.IMG;1	-	-	NO

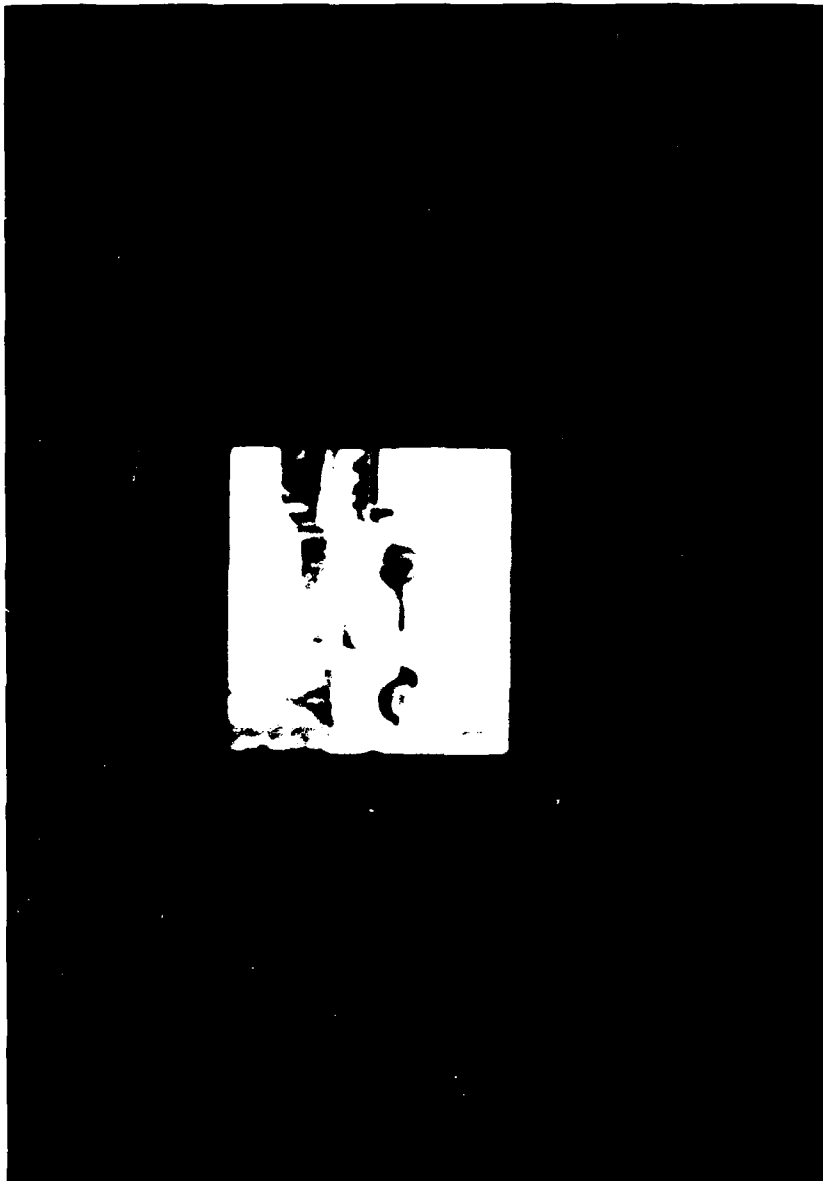


Figure 28. Truck Image (128x128) without Interesting
Points Overwritten (TRUCK.IMG;2)

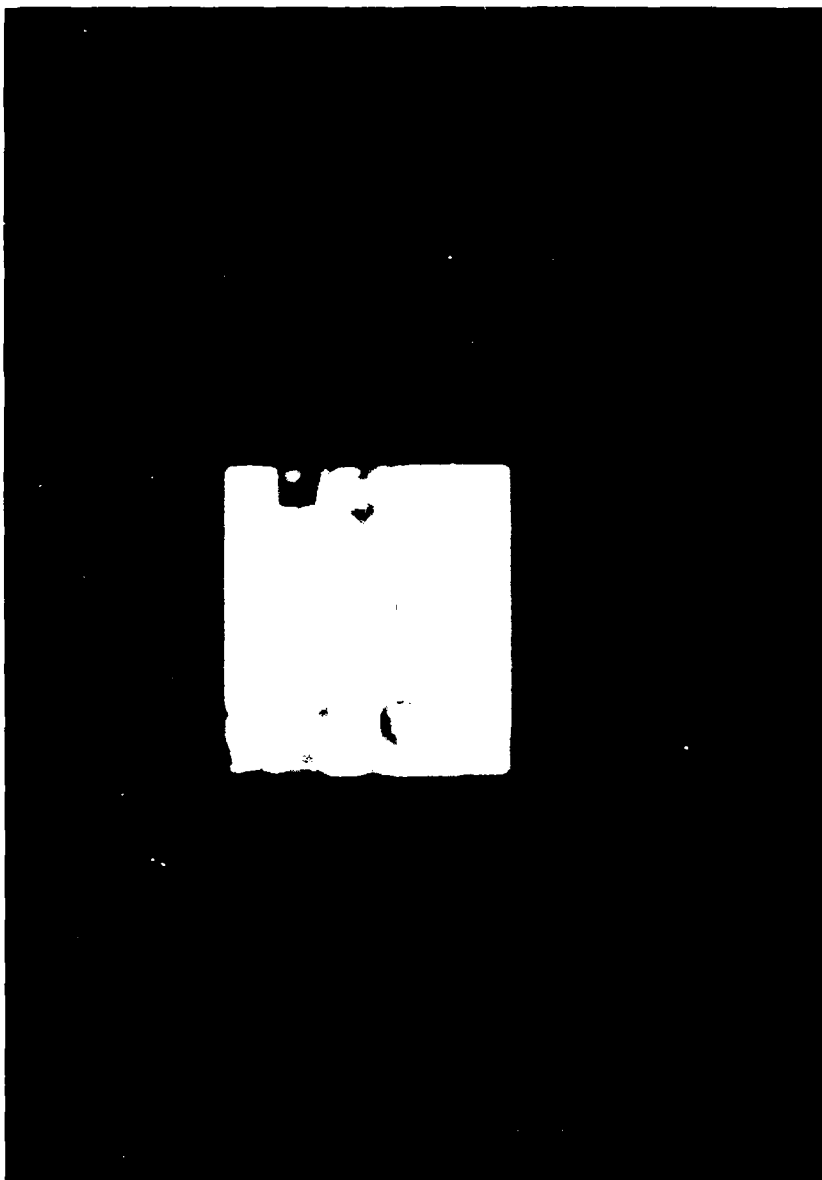


Figure 29. Truck Image (128x128) with 284 Interesting Points Overwritten
(TRUCK_IMG;2) using the Moravec Interest Operator (IOP#2)



Figure 30. Truck Image (TRUCK.IMG;2) Blown Up to 512x512
without Interesting Points Overwritten



Figure 31. Truck Image (TRUCK.IMG;2) Blown Up to 512x512 with 284 Interesting
Points Overwritten using 10p#2 (Moravec Operator)



S

Figure 32. Truck Image (TRUCK.IMG;2) Blown Up to 512x512 with 465 Interesting
Points Overwritten using the Simple Variance Interest Operator (IOP#1)

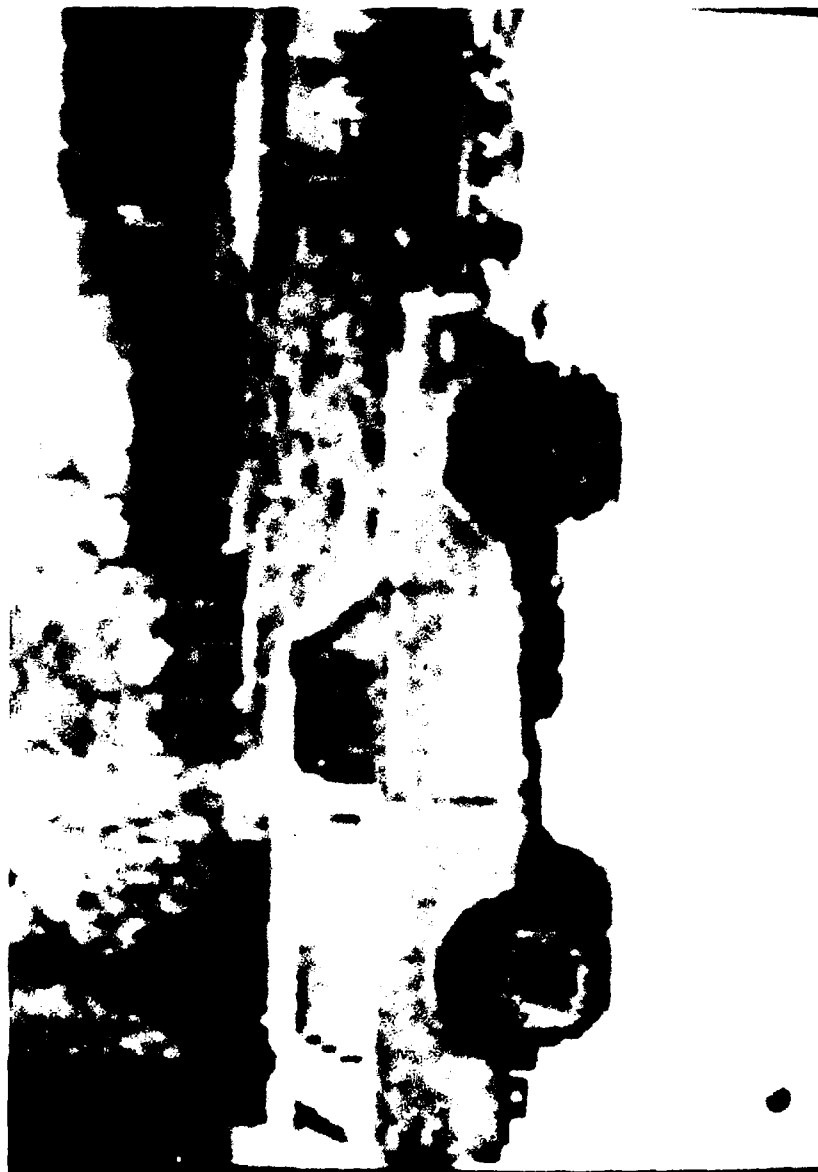


Figure 33. Truck Image (TRUCK.IMG;2) Blown Up to 512x512 with 27 Interesting Points Overwritten using the Edged Variance Interest Operator (IOP#3)

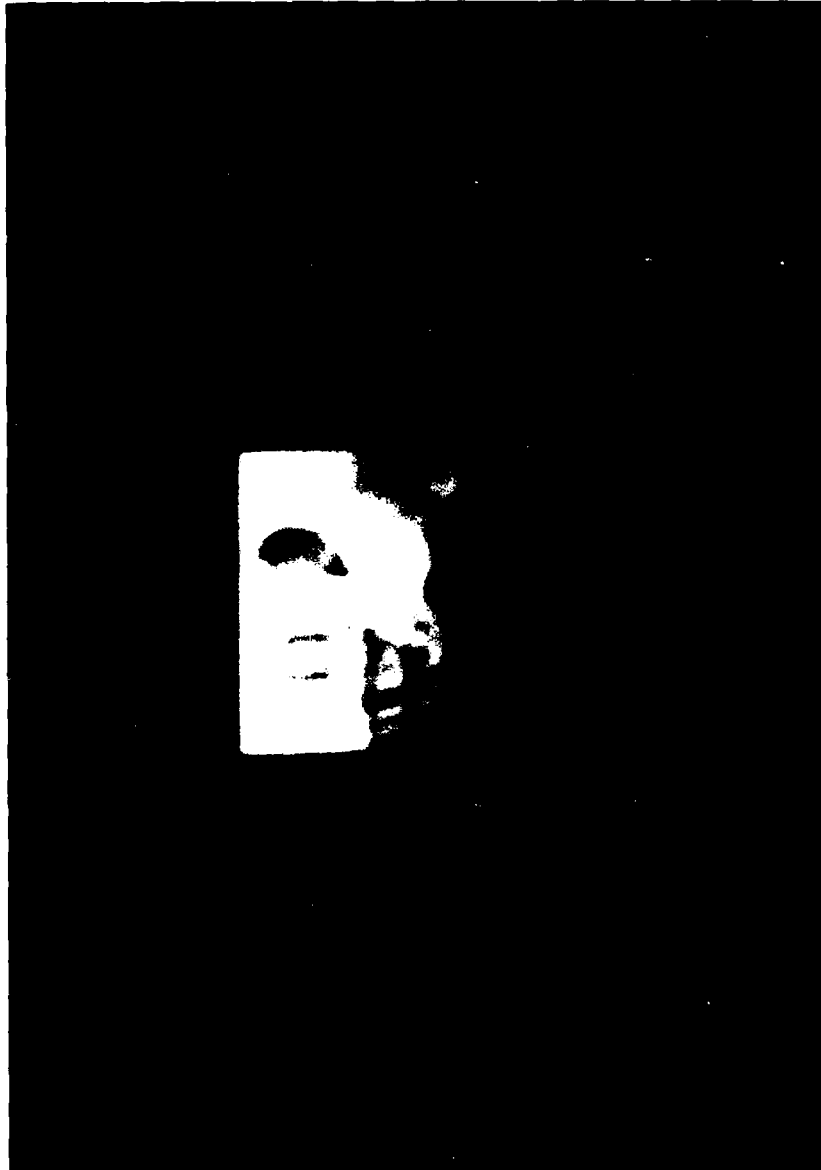


Figure 34. Person Sitting in Chair Image (128x128) without
Interesting Points Overwritten (PERSON.IMG;1)

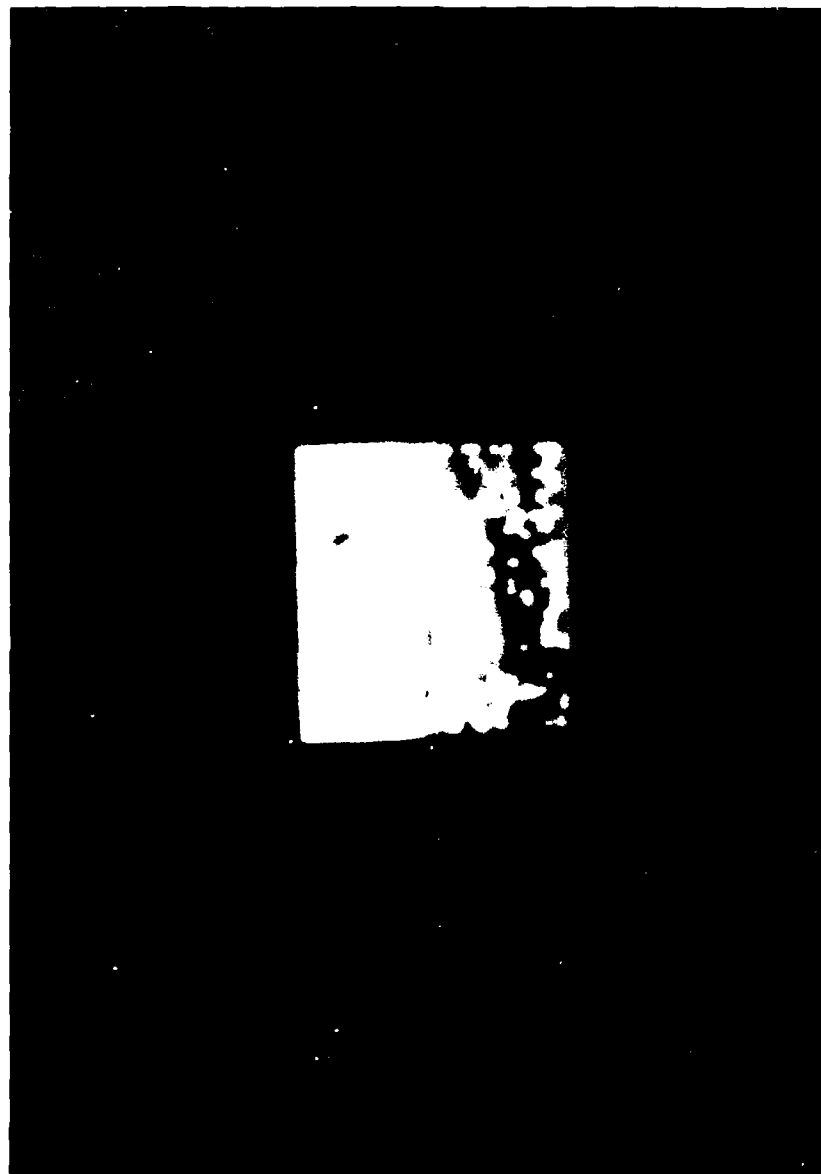


Figure 35. Person Sitting in Chair Image (128x128) with 287 Interesting Points Over-
written (PERSON.IMG;1) using the Moravec Interest Operator (IOP#2)



Figure 36. Person Sitting in Chair Image (PERSON.IMG;1) Blown Up to 512x512
without Interesting Points Overwritten



Figure 37. Person Sitting in Chair Image (PERSON.IMG;1) Blown Up to 512x512 with 287
Interesting Points Overwritten using IOP#2 (Moravec Operator)



Figure 38. Person Sitting in Chair Image (PERSON.IMG:1) Blown Up to 512x512 with 59
Interesting Points Overwritten using the Edged Variance Interest Oper-
ator (IOP#3)

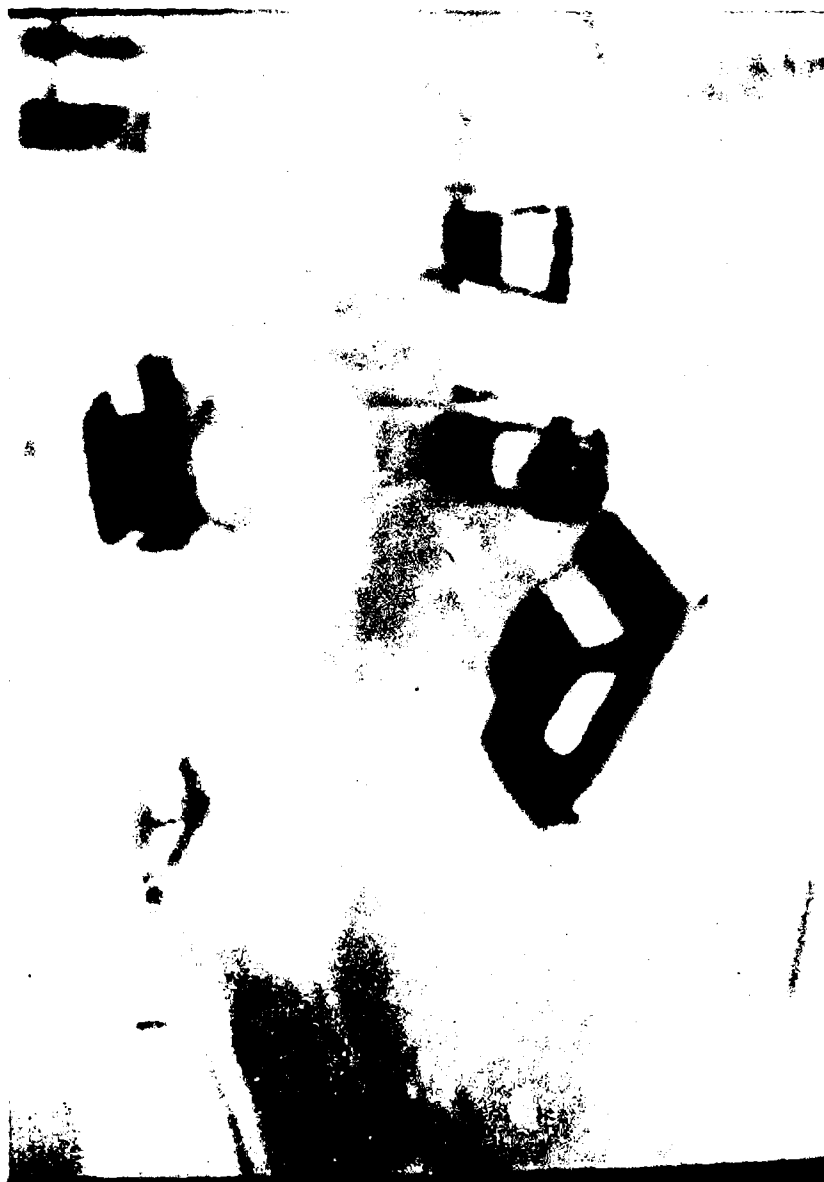


Figure 39. Image of a Taxi Cab Turning a Corner Near the University
of Hamburg (512x512) (TRAFIC.IMG;1)



Figure 40. Image of a Section of the Night Vision Lab Terrain
Board (512x512) (TERRAIN.IMG;1)

image to almost 4 hours for a 512x512 image for the Moravec Operator (IOP#2). This time was about the same for the simple variance operator, and almost double for the edged variance operator.

Next, problems associated with the matching algorithm must also be resolved. The matching efficiencies obtained during this study could not even begin to produce satisfactory results for target tracking and other similar operations. As in the case of the interest operators, the processing speed of the matching algorithm must also be increased before a real-time program can be obtained. Processing times for matching two 128x128 images averaged 5 to 7 minutes. Matching of two 512x512 images was not attempted.

Finally, an integrated system must be developed to incorporate the features of the DIDA program into the DIPS system to provide a completely flexible analytical tool to perform real-time disparity analysis operations, and make modifications to the appropriate program parameters as the need arises. In its completed form, the DIPS processing system should allow the user to manipulate image data files or modify them as the need arises, perform image smoothing, and accomplish real-time disparity analysis of time-varying imagery.

VI. Recommendations for Further Study

The following recommendations for further study are suggested for each section of program DIDA. Following these recommendations, some general recommendations pertaining to the overall effort are given.

Image Operations Recommendations

The following recommendations for further study are suggested for the image operations section of program DIDA:

1. An option to allow the user to change single intensity elements within a given image.
2. An option to allow the user to perform image operations on a magnetic tape file as well as a disk file.
3. An option to allow the user to transfer image files from disk to magnetic tape, or from magnetic tape to disk.

Interest Operator Recommendations

The following recommendations for further study are suggested for the interesting point selection section of program DIDA:

1. A method for inputting sub-images and locating interesting points on them. This entails bringing in only a portion of a complete image file into the program for processing, thereby allowing the user to look at a certain selected area of the image for interesting points. This would be particularly useful if the interesting points of only a certain image feature (such as a tank) were desired.
2. A method for moving the processing window over the image in a non-sequential fashion. A search pattern other than left-to-right, top-to-bottom may be desired for some reason. For instance, a non-sequential window movement might be desired for studying the manner in which a human views a

scene.

3. An adaptive threshold to select interesting points in lieu of being set by the user. This could be accomplished by first finding the variance over the entire image, then, modifying that variance based on the number of interesting points desired to produce the appropriate threshold.
4. An interest operator using binary images (i.e., images represented by only ones and zeros).
5. An interest operator using a blob detection algorithm.
6. An interest operator using "micro-texture". Consider the 2x2 window shown below:

a	b
c	d

Define the following quantities

$$4t_o = a+b+c+d \quad (69)$$

$$4t_x = -a+b-c+d \quad (70)$$

$$4t_y = a+b-c-d \quad (71)$$

$$4t_z = -a+b-c+d \quad (72)$$

In the above expressions, t_o corresponds to the average over the window, t_x and t_y are gradient vectors, and t_z is an operator similar in function to a 2nd derivative operation. Testing for t_z greater than some specified threshold produces the interest operator. For areas larger than 2x2, a Hadamard Transform may be employed to look at functions of the combinations of t_o , t_x , t_y , and t_z for each 2x2 window.

7. Interesting points should be selected for varying processing window sizes to determine the effect of window size on placement of interesting points on the image being processed.
8. Various values of epsilon should be tried for the maximum directed variance tests for interest operator #2 to determine their effect on interesting point selection.

Matching Algorithm Recommendations

The following recommendations for further study are suggested for the interesting point matching section of program DIDA.

1. A method of accurately determining matched points after matching is complete. At the end of the matching process, a list of the points in interesting point set 1 and their corresponding matching point in interesting point set 2 is given. A method is needed for verifying the accuracy of these matches.
2. A means for treating ambiguous nodes. When the matching process is complete, some nodes may have several labels with finite non-zero probabilities. It may be possible to examine these ambiguous nodes to more accurately determine if a match exists.
3. A set of criteria for matching. In this effort, the matching efficiency was defined as the ratio of the number of matched to unmatched points expressed as a percentage. This did not account for points which were incorrectly matched, or ambiguous nodes. A way of accurately defining the matching efficiency of each interest operator, as well as other pertinent matching parameters, should be described.
4. The effect of correlation window size on initial probabilities. Correlation windows of various sizes (not necessary equal) should be tried to determine their effect on the initial probabilities computed from them for each of the six weighting algorithms.
5. The effect of the weighting algorithm parameter, C, on the initial probabilities. For the same correlation window size each time, the parameter, C, in each of the weighting algorithms should be varied to determine its effect on the initial probabilities computed.
6. Test weighting algorithms for rotational invariance. With the two images rotated with respect to one another, the ability of each weighting algorithm to assign meaningful initial probabilities should be assessed for various angles of rotation.
7. Describe weighting algorithms which are rotationally invariant, and produce a better set of initial

probabilities. The possibility of using Fourier Transforms, Horev filtering, and other similar operations should be examined.

8. The effect of the size of the consistency neighborhood on the number of points that are matched.
9. The incorporation of a coefficient in the consistency factor equations (equations (41) and (42)) to account for the distance between nodes within the consistency neighborhood. In this study, this coefficient was 1 for all nodes within the neighborhood. However, in general, equations (41) and (42) can be written as

$$q_i^k(l) = \sum_{\substack{j \text{ near } a_i \\ j \neq i}} \{ ||\Sigma l' || \leq T \} r_{ij} p_j^k(l') \quad (l \neq l^*) \quad (73)$$

and, for l^*

$$q_i^k(l^*) = \sum_{\substack{j \text{ near } a_i \\ j \neq i}} r_{ij} p_j^k(l^*) \quad (74)$$

In the above equations, r_{ij} is the coefficient which relates the distance between nodes within the consistency neighborhood to the consistency factor, q . One possible assessment of r_{ij} might be

$$r_{ij} = [(x_i - x_j)^2 + (y_i - y_j)^2]^{1/2} \quad (75)$$

10. The effect of changes of the similarity parameter, T , on the consistency factor, q . This should also be related to the resulting matching efficiency.
11. The effect of changes of the parameters A and B on the probability updates. Each parameter should be varied independently, and the resulting updated probabilities noted.

Image Display Recommendations

AD-A127 409

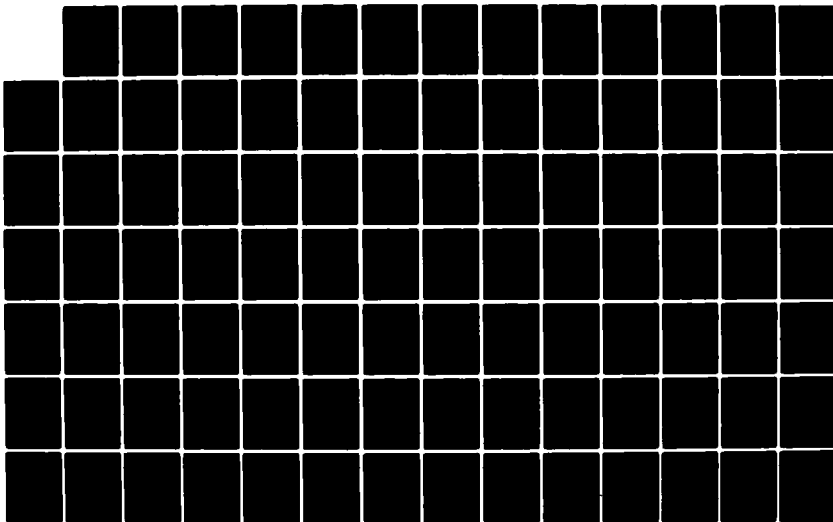
DISPARITY ANALYSIS OF TIME-VARYING IMAGERY(U) AIR FORCE
INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF
ENGINEERING F D COOPER DEC 81 AFIT/GEO/EE/81D-1

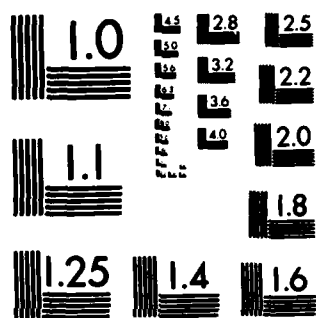
3/4

UNCLASSIFIED

F/G 20/6

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

The following recommendations for further study are suggested for the image display section of program DIDA.

1. An inverse image capability for displaying both normal images and their negatives.
2. A method for displaying images in their natural color.
3. A method for writing text on or below a displayed image.
4. An option for interactively changing the video look-up table of the RAMTEK driver to produce "pseudo-color" images with a user-specified color versus intensity distribution.
5. A method for overwriting the interesting points on a displayed image in different colors based on the corresponding interest operator. For instance, interesting points selected using interest operator # 1 could be overwritten in green, those from interest operator #2 in blue, and those from interest operator #3 in red. Those interesting points which overlap will then be displayed in a different color depending on which points are overlaid.
6. A method of displaying split-screen images for comparing the results of the matching algorithm.

General Recommendations

The following recommendations for further study are suggested for the improvement of the overall structure of program DIDA.

1. A method for processing the entire image at one time. Having access to the entire image in memory all at one time greatly enhances both processing speed and flexibility of processing operations. For instance, it would be extremely difficult to employ a non-sequential processing window movement when the image must be buffered into core memory several rows at a time.
2. The use of array processors to increase processing speed.
3. Rewriting the program in assembly language to increase processing speed.

4. Integration of program DIDA into the DIPS program.
5. Development of a real-time display algorithm to demonstrate the real-time disparity analysis algorithm.

Bibliography

1. Rome Air Development Center. Image Processing System Software. Programming Manual. Buffalo, N.Y.: Amherst Systems, Inc., November 1978.
2. Rome Air Development Center. Image Processing System Software. User's Manual. Buffalo, N.Y.: Amherst Systems, Inc., November 1978.
3. Choi, Dug K. Image Processing System Software for Dynamic Image Disparity Analysis. Unpublished Technical Report. AFWAL/AAAT-1, Avionics Laboratory, Wright-Patterson Air Force Base, Ohio, May 1981.
4. Barnard, Stephen T. and William B. Thompson. "Disparity Analysis of Images," IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-2 (4): 333-340 (July 1980).
5. Jain, R., et al. "Segmentation through the Detection of Changes Due to Motion," Computer Graphics and Image Processing, 11: 13-34 (February 23, 1979).
6. Thompson, William B. "Combining Motion and Contrast for Segmentation," IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-2 (6): 543-549 (November 1980).
7. Softech. Structured Analysis and Design Technique. An Introduction to SADT. Waltham, MA.: Softech, Inc., November 1976.
8. Jensen, Randall W. and Charles C. Tonies. Software Engineering. New Jersey: Prentice-Hall, Inc., 1979.
9. Hannah, Marsha J., et al. Passive Imagery Navigation. Palo Alto, CA.: Lockheed Palo Alto Research Laboratory, December 1980.
10. Snyder, Wesley E. "Computer Analysis of Time-Varying Images," Computer, 18: 7-9 (August 1981).
11. Duda, Richard O. and Peter E. Hart. Pattern Classification and Scene Analysis. New York: John Wiley and Sons, Inc., 1973.
12. Hannah, Marsha J. "Bootstrap Stereo," Proceedings of the Image Understanding Workshop. 201-208. College Park, MD, April 1980.
13. Moravec, Hans P. "Towards Automatic Visual Obstacle Avoidance," Proceedings of the 5th International Joint Conference on Artificial Intelligence, II (1): 584 (August 1977).
14. Gennery, Donald B. "A Stereo Vision System for an Autonomous Vehicle," Proceedings of the 5th International Joint Conference on Artificial Intelligence, II (1): 576-582 (August 1977).

15. Martin, William N. and Jake K. Aggarwal. "Survey of Dynamic Scene Analysis," Computer Graphics and Image Processing, 7(1): 356-374 (April 1977).
16. Mendenhall, William and Richard L. Scheaffer. Mathematical Statistics with Applications. Belmont, CA.: Wadsworth Publishing Company, 1973.
17. Dougherty, Llewellyn S. New Techniques for Tracking Sequences of Digitized Images. PHD Thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, October 1978.
18. Thompson, William B. and Stephen T. Barnard. "Lower-level Estimation and Interpretation of Visual Motion," Computer, 18: 20-28 (August 1981).
19. Zucker, Stephen W., et al. "Continuous Relaxation and Local Maxima Selection: Conditions for Equivalence," IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-3 (2): 117-127 (March 1981).
20. Danker, Alan J. and Azriel Rosenfeld. "Blob Detection by Relaxation," IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-3 (1): 79-92 (January 1981).
21. Maybeck, Peter S. Stochastic Models, Estimation, and Control. New York: Academic Press, Inc. 1979.

Appendix A: Glossary of Symbols, Variables, and Arrays

<u>NAME</u>	<u>ASSOCIATED BLOCK COMMON</u>	<u>BRIEF DESCRIPTION OF FUNCTION</u>
IMGROW	/IMAGE/	Number of image rows
IMGCOL		Number of image columns
IMGSZ		Product of image rows and columns
IMGINV		Reciprocal of image size
LUN	/FILE/	Logical unit number associated with image file
NCHAR		Number of characters in file name
FNAME(34)		Device/UIC/filename of image file
IOBUFR(L,M)		Buffer byte data to/from image file
INTBUF(M)	/INBUF/	Store converted byte data in 16-bit word
WNDROW	/WINDOW/	Number of window rows
WNDCOL		Number of window columns
WNDRWH		WNDROW/2
WNDCLH		WNDROW/2
WNSZ	-	Product of window rows and columns
WINDIN		Reciprocal of window size
STORE(N,M)		VIRTUAL generic storage array
CWDROW		Number of correlation window rows
CWDCOL	/CORWND/	Number of correlation window columns
CWDRWH		CWDROW/2+1
CWDCLH		CWDCOL/2+1
CWDSZ		Product of correlation window rows and column
CWDIN	/IOP/	Reciprocal of correlation window size
IOPNBR		Number of interest operator used
IPFLAG		Interesting point storage flag
LUNCW		Logical unit number of correlation window with scratch file
LUNI	/IFILE1/	Logical unit number associated with interesting point set file
NCHAR1		Number of characters in file name
FNAME1(34)		Device /UIC/ filename of interesting point set file
IPKNT		Number of interesting points selected
IPR(M)	/IPS/	Array of interesting point image row locations
IPC(M)		Array of interesting point image column locations
VALUIP(M)		Value of each interesting point selected
IPCWND(M/3)		Buffer array for correlation window storage and retrieval
IPKNT1	/IPS1/	Number of interesting points in first set to be matched.
IPTYPI		Interest operator used to select interesting

CWROW1		points
CWCOL1		Number of rows of IP correlation windows
CWSZE1		Number of columns of IP correlation windows
		Product of correlation window rows and columns
LUN2	/IFILE2/	Logical unit number associated with second interesting point set file
NCHAR2		Number of characters in file name
FNAME2(34)		Device /UIC/ filename of interesting point set file
IPKNT2	/IPS2/	Number of interesting points in second set to be matched
IPTYP2		Interest operator used to select interesting points
CWROW2		Number of rows of IP correlation windows
CWCOL2		Number of columns of IP correlation windows
CWSZE2		Product of correlation window rows and columns
THRESH	-	Variance threshold for interesting point tests
VARNCE	-	Value of simple variance computed in IOPl
COLSUM	/MEAN/	Sum of intensity elements for window position
SQRSUM		Sums of squares of intensities over window
MEAN		Value of mean computed in MEANFD
IMGEND	-	End-of-image flag set in WNDMOV
ROWKNT	/MOVE/	Top row of window WRT image
COLKNT		Left column of window WRT image
ROWDIF		Bottom of window WRT image
COLDIF		Right column of window WRT image
S(M)	/REG1/	Intensity element partial sums
T(M)		Partial sums of squares of intensity elements
WNDKNT	/COUNT/	Intensity array row change pointer
BUFKNT		Input buffer array row change pointer
RECNUM		Value of current record being processed
IPROW	-	Row location of each interesting point
IPCOL	-	Column location of each interesting point
VALUE	-	Value of each interesting point
CWDLCL	-	Variable to test for left column of image
CWDTRW	-	Variable to test for top row of image
CWDRCL	-	Variable to test for right column of image
CWDBRW	-	Variable to test for bottom row of image
CWDCLS	-	Correlation window column starting position
CWDRWS	-	Correlation window row starting position
CWDCLE	-	Correlation window column end position
CWDRWE	-	Correlation window row end position
IPCKNT	-	Initial value of correlation window element
IMCLST	-	Relative location of image data in input records
IBUF(M)	/BUF/	Correlation window record input buffer
EPSLON	/DSTOR/	Maximum directed variance test threshold

DCOL		Number of directed variance values per D row.
D(2,M)		Storage array for directed variance values
XSUM	/DRVAR/	Sum of row squared intensity differences
YSUM		Sum of column squared intensity differences
ASUM		Upper-to-lower diagonal sum
BSUM		Lower-to-upper diagonal sum
X(M)	/REG2/	Partial sums of row squared differences
Y(M)		Partial sums of columns squared differences
A(M)		Upper-to-lower diagonal partial sums
B(M)		Lower-to-upper diagonal partial sums
DIFROW	-	Elemental row difference
DIFCOL	-	Elemental column difference
DIFULD	-	Upper-to-lower diagonal difference
DIFLUD	-	Lower-to-upper diagonal difference
DFSQRW	-	DIFROW/2
DFSQCL	-	DIFCOL/2
DFSQUL	-	DIFULD/2
DFSQLU	-	DIFLUD/2
DFCLED	-	Difference of last column intensities
DFSQCE	-	DFCLED/2
DVAR1	-	Row directional variance
DVAR2	-	Column directional variance
DVAR3	-	Upper-to-lower diagonal directional variance
DVAR4	-	Lower-to-upper diagonal directional variance
DIRVAR	-	Overall directed variance value
EDGVAR	-	Edged variance value
EFLAG	/EDVAR/	Edged variance flag
RTOMIN		Minimum of directional variance ratios
RATIO1	-	Ratio of DVAR1 to DVAR2
RATIO2	-	Ratio of DVAR2 to DVAR1
RATIO3	-	Ratio of DVAR3 to DVAR2
RATIO4	-	Ratio of DVAR4 to DVAR3
DWDROW	/DSPWND/	Number of disparity window rows
DWDCOL		Number of disparity window columns
DWDRWH		DWDROW/2
DWDCLH		DWDCOL/2
DWDSZE		Product of disparity window rows and columns
DWDINV		Reciprocal of disparity window size
CWTEST	-	Specifies the orientation of the correlation windows
W8ANBR	-	Number of weighting algorithm selected
C	-	Arbitrary constant used in weighting algorithms
NGHROW	/NBHOOD/	Number of consistency neighborhood rows
NGHCOL		Number of consistency neighborhood columns
NGHRWH		NGHROW/2
NGHCLH		NGHCOL/2
NGHSZE		Product of neighborhood rows and columns
NGHINV		Reciprocal of neighborhood size
A	-	Damping parameter for consistency factor

B	-	Gain parameter for consistency factor
T	-	Label consistency threshold
LBLTST	-	Maximum label storage position
IPROW1	-	X locations of interesting point set 1
IPCOL1	-	Y locations of interesting point set 1
LX	-	X disparity label value
LY	-	Y disparity label value
W	-	Weight returned by weighting algorithm
LBLKNT	-	Number of labels
IPROW2	-	X locations of interesting point set 2
IPCOL2	-	Y locations of interesting point set 2
DRWFLG	-	Disparity window row flag
DCLFLG	-	Disparity window column flag
DRWMIN	-	Lower bound on disparity window row
DRWMAX	-	Upper bound on disparity window row
DCLMIN	-	Lower bound on disparity window column
DCLMAX	-	Upper bound on disparity window column
IRWDIF	-	Row difference of correlation windows
IRWDFH	-	IRWDIF/2
ICLDIF	-	Column difference of correlation windows
ICLDFH	-	ICLDIF/2
IPCW1(M/3)	/REG/	Correlation window storage for IP set 1
IPCW2(M/3)		Correlation window storage for IP set 2
ITMP1(M/3)	/REG2/	Temporary correlation window storage for IP set 1
ITMP2(M/3)		Temporary correlation window storage for IP set 2
CWNDIF	-	Difference of correlation window pixels
DIFSQR	-	CWNDIF/2
SUM	-	Sum of DIFSQR over window area
SUM1	-	Sum of elements of correlation window 1
SUM2	-	Sum of elements of correlation window 2
CWAVE1	-	Average of SUM1
CWAVE2	-	Average of SUM2
ASUM1	-	Upper left coordinate sum for window 1
BSUM1	-	Upper right coordinate sum for window 1
CSUM1	-	Lower left coordinate sum for window 1
DSUM1	-	Lower right coordinate sum for window 1
ASUM2	-	Upper left coordinate sum for window 2
BSUM2	-	Upper right coordinate sum for window 2
CSUM2	-	Lower left coordinate sum for window 2
DSUM2	-	Lower right coordinate sum for window 2
A1	-	Average of ASUM1
B1	-	Average of BSUM1
C1	-	Average of CSUM1
D1	-	Average of DSUM1
A2	-	Average of ASUM2
B2	-	Average of BSUM2
C2	-	Average of CSUM2
D2	-	Average of DSUM2

NDF	-	Displacement between coordinate sums
G1	-	Gradient of correlation window 1
G2	-	Gradient of correlation window 2
ISQRT	-	Size parameter of window in W8ALG4
SQRSM1	-	Sum-of-squares of correlation window 1
SQRSM2	-	Sum-of-squares of correlation window 2
CMEAN1	-	Mean of correlation window 1
CMEAN2	-	Mean of correlation window 2
SQRCM1	-	Square of mean of correlation window 1
SQRCM2	-	Square of mean of correlation window 2
CWVAR1	-	Variance of correlation window 1
CWVAR2	-	Variance of correlation window 2
CVRDIF	-	Difference of variances of correlation windows
CRSPRD	-	Product of cross terms of correlation windows
CPSUM	-	Sum of cross products
SQRCS1	-	Square of sums for correlation window 1
SQRCS2	-	Square of sums for correlation window 2
C12	-	Covariance of both correlation windows
V1	-	Variance of correlation window 1
V2	-	Variance of correlation window 2
WSUM	-	Sum of weights for each node
WMAX	-	Maximum weight of each node
CPROB	-	Conditional probability of a label of ai
NRWMIN	-	Lower bond on consistency neighborhood rows
NRWMAX	-	Upper bond on consistency neighborhood rows
NCLMIN	-	Lower bond on consistency neighborhood columns
NCLMAX	-	Upper bond on consistency neighborhood columns
NROW1	-	Row coordinate of node ai
NCOL1	-	Column coordinate of node ai
NROW2	-	Row coordinate of node aj
NCOL2	-	Column coordinate of node aj
QL(1)	-	Match consistency factor for label 1
QLSTR	-	No-match consistency factor
NRWFLG	-	Neighborhood row flag
NCLFLG	-	Neighborhood column flag
LBLRDF	-	Label row difference
LBLCDF	-	Label column difference
LBRFLG	-	Label row flag
LBCFLG	-	Label column flag
PSUM	-	Sum of updated probabilities
LUNRM	-	Logical unit number of RAMTEK driver
IPAR(6)	/RBUFF/	Initialization of RAMTEK driver
IOSTAT(2)	-	Initialization of RAMTEK driver
IVLT(256)	-	RAMTEK video look-up table
IMBUF(M)	-	Buffer array for RAMTEK driver
ICOLOR	-	Color flag for image displays

ISCALE

-

Scaling flag for image displays

Appendix B: DIDA Control Program

This appendix contains the FORTRAN source listing for the main control program of program DIDA.

```
PROGRAM DIDA
C
C PROGRAM DIDA CONTROLS THE OPERATION OF THE DYNAMIC IMAGE
C DISPARITY ANALYSIS ROUTINES. THE USER IS ASKED TO SELECT
C THE DESIRED OPERATION. THE SUBROUTINES REQUIRED TO
C PERFORM THAT OPERATION ARE THEN CALLED. AFTER PROCESSING
C IS COMPLETE, THE USER IS ASKED TO EITHER CONTINUE PROGRAM
C EXECUTION, OR TERMINATE.
C
C *** COMMON BLOCKS AND INITIALIZATION ***
C
C INCLUDE 'COMMON.FTN'
C
C *** VIRTUAL STORAGE AREA ***
C
C VIRTUAL STORE(N,M)
C
C *** BUFFER AREA ***
C
C COMMON /IBUF/ IBUF(M)
C COMMON /BUF/ IOBUFR(L,M)
C COMMON /INBUF/ INTBUF(M)
C
C *** REGISTER AREA ***
C
C COMMON /REG1/ S(M),T(M)
C COMMON /REG2/ X(M),Y(M),A(M),B(M)
C
C *** FILE AREA ***
C
C COMMON /CWFILE/ LUNCW
C COMMON /FILE/ LUN,NCHAR,FNAME(34)
C COMMON /IFILE1/ LUN1,NCHAR1,FNAME1(34)
C COMMON /IFILE2/ LUN2,NCHAR2,FNAME2(34)
C
C *** INTERESTING POINT SET STORAGE AREA ***
C
C COMMON /IPS/ IPKNT,IPR(M),IPC(M),VALUIP(M),IPWND(M/3)
C COMMON /IPS1/ IPKNT1,IPTYP1,CWROW1,CWCOL1,CWSZE1
C COMMON /IPS2/ IPKNT2,IPTYP2,CWROW2,CWCOL2,CWSZE2
C
C *** INTEREST OPERATOR VARIABLES AREA ***
```

```

C      COMMON /IOP/ IOPNBR,IPFLAG
      COMMON /MEAN/ COLSUM,SQRSUM,MEAN
      COMMON /DSTOR/ DCOL,EPSLON,D(2,M)
      COMMON /DRVAR/ XSUM,YSUM,ASUM,BSUM
      COMMON /EDVAR/ EFLAG,RTOMIN

C      *** WINDOW VARIABLES AREA ***
C
      COMMON /COUNT/ WNDKNT,BUFKNT,RECNUM
      COMMON /MOVE/ ROWKNT,COLKNT,ROWDIF,COLDIF
      COMMON /IMAGE/ IMGROW,IMGCOL,IMGSZE,IMGINV
      COMMON /WINDOW/ WNDROW,WNDROWH,WNDCLH,WNDSZE,WNDINV
      COMMON /CORWND/ CWDROW,CWDCOL,CWDRWH,CWDCLH,CWDSZE,CWDINV
      COMMON /DSPWND/ DWDROW,DWDCOL,DWDRWH,DWDCLH,DWDSZE,DWDINV
      COMMON /NBHOOD/ NGHROW,NGHCOL,NGHRWH,NGHCLH,NGHSZE,NGHINV

C      LOGICAL*1 IBUF,IOBUFR,FNAME,FNAME1,FNAME2
C
C      *** INTEGER VARIABLES ***
C
      INTEGER END,ERROR,OPRNR,CWROW1,CWCOL1,CWSZE1
      INTEGER CWROW2,CWCOL2,CWSZE2,DCOL,EFLAG,WNDKNT
      INTEGER BUFKNT,RECNUM,ROWKNT,COLKNT,ROWDIF,COLDIF
      INTEGER WNDROW,WNDROWH,WNDCLH,WNDSZE
      INTEGER CWDROW,CWDCOL,CWDRWH,CWDCLH,CWDSZE
      INTEGER DWDROW,DWDCOL,DWDRWH,DWDCLH,DWDSZE

C      *** REAL VARIABLES ***
C
      REAL IMGSZE,IMGINV,NGHINV,MEAN

C      *** INITIALIZATION ***
C
      TYPE *, ' '
      TYPE *, ' '
      TYPE *, 'DO YOU WANT THE PROGRAM OUTPUT TO BE ROUTED TO THE'
      TYPE *, 'LINE PRINTER OR SYSTEM TEMPORARY FILE?'
      TYPE *, ' '
      TYPE *, 'IF YOU WANT THE PROGRAM OUTPUT TO GO TO THE LINE'
      TYPE *, 'PRINTER, TYPE YES. OTHERWISE, TYPE NO. TYPING NO'
      TYPE *, 'CAUSES THE PROGRAM OUTPUT TO BE SENT TO THE SYSTEM'
      TYPE *, 'FILE NAME:'
      TYPE *, ' '
      TYPE *, 'SY:DIDAOUT.TMP'
      TYPE *, ' '
      TYPE 11
      ACCEPT 12,ANSWER
      CALL ANSCHK(ANSWER)
      IF(ANSWER.EQ.'YES')GO TO 2

```

```

CALL ASSIGN(6,'SY:DIDAOUT.TMP')
GO TO 3
2 CALL ASSIGN(6,'LP:')
3 ERROR = 0
END = 0
C
WRITE (6,*) '*** BEGIN DISPARITY ANALYSIS PROGRAM EXECUTION ***'
WRITE (6,*) ' '
WRITE (6,*) ' '
1 TYPE *, ' '
TYPE *, ' '
TYPE *, 'ENTER THE NUMBER CORRESPONDING TO THE OPERATION'
TYPE *, 'YOU WISH TO PERFORM:'
TYPE *, ' '
TYPE *, ' 1. PERFORM IMAGE OPERATIONS'
TYPE *, ' 2. FIND INTERESTING POINTS'
TYPE *, ' 3. MATCH INTERESTING POINTS'
TYPE *, ' 4. DISPLAY IMAGES ON A CRT'
TYPE *, ' 5. TERMINATE PROGRAM EXECUTION'
TYPE *, ' '
TYPE 5
5 FORMAT(5X,'OPERATION NUMBER = ',S)
ACCEPT *,OPRNBR
IF(OPRNBR.LT.1.OR.OPRNBR.GT.5)GO TO 50
GO TO(10,20,30,40,200)OPRNBR
10 CALL IMGOPR(ERROR,END)
IF(ERROR.EQ.1)GO TO 100
IF(END.EQ.1)GO TO 200
TYPE *, ' '
TYPE *, 'DO YOU WISH TO PERFORM OTHER IMAGE OPERATIONS?'
TYPE *, 'IF YOU DO, TYPE YES. OTHERWISE, TYPE NO.'
TYPE *, ' '
TYPE 11
11 FORMAT(5X,'ANSWER = ',S)
ACCEPT 12,ANSWER
12 FORMAT(A3)
CALL ANSCHK(ANSWER)
IF(ANSWER.EQ.'YES')GO TO 10
WRITE (6,*) '*** IMAGE OPERATIONS COMPLETED ***'
GO TO 60
20 CALL INTPTS(STORE,ERROR,END)
IF(ERROR.EQ.1)GO TO 100
IF(END.EQ.1)GO TO 200
TYPE *, ' '
TYPE *, 'DO YOU WISH TO FIND INTERESTING POINTS ON ANOTHER IMAGE?'
TYPE *, 'IF YOU DO, TYPE YES. OTHERWISE, TYPE NO.'
TYPE *, ' '
TYPE 11
ACCEPT 12,ANSWER
CALL ANSCHK(ANSWER)

```

```

IF(ANSWER.EQ.'YES')GO TO 20
WRITE (6,*) '*** INTERESTING POINT PROCSSING COMPLETE ***'
GO TO 60
30 CALL IPNTCH(STORE,ERROR,END)
IF(ERROR.EQ.1)GO TO 100
IF(END.EQ.1)GO TO 200
TYPE *, ' '
TYPE *, 'DO YOU WISH PERFORM MATCHING ON OTHER IMAGES?'
TYPE *, 'IF YOU DO,TYPE YES. OTHERWISE, TYPE NO.'
TYPE *, ' '
TYPE 11
ACCEPT 12,ANSWER
CALL ANSCHK(ANSWER)
IF(ANSWER.EQ.'YES')GO TO 30
WRITE (6,*) '*** INTERESTING POINT MATCHING COMPLETE ***'
GO TO 60
40 CALL IMGDSP(ERROR,END)
IF(ERROR.EQ.1)GO TO 100
IF(END.EQ.1)GO TO 200
TYPE *, ' '
TYPE *, 'DO YOU WISH TO DISPLAY OTHER IMAGES?'
TYPE *, 'IF YOU DO, TYPE YES. OTHERWISE, TYPE NO.'
TYPE *, ' '
TYPE 11
ACCEPT 12,ANSWER
CALL ANSCHK(ANSWER)
IF(ANSWER.EQ.'YES')GO TO 40
WRITE (6,*) '*** IMAGE DISPLAY OPERATIONS COMPLETED ***'
GO TO 60
50 TYPE *, ' '
TYPE *, '*** INVALID OPERATION NUMBER',OPRNR,' ATTEMPTED ***'
60 TYPE *, ' '
TYPE *, 'DO YOU WISH TO TERMINATE PROGRAM EXECUTION?'
TYPE *, 'IF YOU DO, TYPE YES. OTHERWISE, TYPE NO.'
TYPE *, ' '
TYPE 11
ACCEPT 12,ANSWER
CALL ANSCHK(ANSWER)
IF(ANSWER.EQ.'YES')GO TO 200
GO TO 1
100 WRITE (6,*) ' '
WRITE (6,*) '*** EXIT DIDA - ERROR TERMINATION ***'
STOP '*** EXIT DIDA - ERROR TERMINATION ***'
200 WRITE (6,*) ' '
WRITE (6,*) ' '
WRITE (6,*) '*** END DISPARITY ANALYSIS PROGRAM EXECUTION ***'
STOP '*** NORMAL TERMINATION ***'
END

```


File: 'COMMON.FTN' - this file specifies the dimensions of all arrays
in Program DIDA. It includes the single statement:

PARAMETER L=4, M=512, N=32

Appendix C: Image Operations Section

This appendix contains the FORTRAN source listing for the Image Operations Section of program DIDA.

```

      SUBROUTINE IMGOPR(IMGERR,IMGEND)
C
C      THIS SUBROUTINE IS USED TO PERFORM IMAGE OPERATIONS.  THE
C      IMAGE SIZE IS DEFINED FIRST.  THEN THE DESIRED IMAGE OPER-
C      ATION IS SELECTED, AND THAT OPERATION BEGINS.
C
C      WRITE (6,*) '*** ENTER IMGOPR ***'
C
C      *** DEFINE THE IMAGE SIZE ***
C
C      CALL IMGSZE
C
C      *** SELECT THE DESIRED IMAGE OPERATION ***
C
C      CALL IMOSL(IMGERR,IMGEND)
C
      RETURN
      END
```

```

SUBROUTINE IMGSE
C
C THIS SUBROUTINE ALLOWS THE USER TO DETERMINE THE NUMBER OF
C ROWS AND COLUMNS OF THE IMAGE TO BE PROCESSED. FROM THIS,
C THE IMAGE SIZE AND ITS INVERSE MAY BE OBTAINED. THE USER
C IS ASKED TO CHECK THE INPUT INFORMATION AND MAKE CORRECTIONS
C IF NECESSARY.
C
COMMON /IMAGE/ IMGROW,IMGCOL,IMSIZE,IMGINV
REAL IMSIZE,IMGINV
TYPE *,' '
TYPE *,'PLEASE TYPE THE NUMBER OF ROWS AND COLUMNS'
TYPE *,'OF THE IMAGE:'
TYPE *,' '
C
C *** ENTER THE NUMBER OF ROWS AND COLUMNS OF THE IMAGE ***
C
CALL IRCFND
C
C *** COMPUTE THE IMAGE SIZE (NUMBER OF ROWS X NUMBER OF COLUMNS)***
C
IMSIZE = FLOAT(IMGROW)*FLOAT(IMGCOL)
C
C *** COMPUTE THE INVERSE OF THE IMAGE SIZE ***
C
IMGINV = 1.0/IMSIZE
C
WRITE (6,*) ' '
WRITE (6,*) '*** IMAGE INFORMATION ***'
WRITE (6,*) ' '
WRITE (6,*) 'NUMBER OF IMAGE ROWS = ',IMGROW
WRITE (6,*) 'NUMBER OF IMAGE COLUMNS = ',IMGCOL
WRITE (6,*) 'IMAGE SIZE (ROWSXCOLUMNS) = ',IMSIZE
WRITE (6,*) 'INVERSE OF IMAGE SIZE = ',IMGINV
WRITE (6,*) ' '
RETURN
END

```

```

SUBROUTINE IRCFND
C
C THIS SUBROUTINE ALLOWS THE USER TO ENTER THE NUMBER OF ROWS AND
C COLUMNS INTERACTIVELY FOR THE IMAGE TO BE INPUT OR OUTPUT. THE
C USER MAY CORRECT THIS INFORMATION IF AN ENTRY ERROR OCCURS.
C
C *** ENTER THE NUMBER OF ROWS AND COLUMNS ***
C
CALL IMROW
CALL IMCOL
C
C *** OPTIONAL ENTRY INFORMATION CHECK ***
C
TYPE *, ' '
TYPE *, 'DO YOU WISH TO CHECK FOR ENTRY ERRORS?'
TYPE *, 'IF YOU DO, TYPE YES. OTHERWISE, TYPE NO.'
TYPE *, ' '
TYPE 1
1  FORMAT(5X, 'ANSWER = ', $)
   ACCEPT 2, ANSWER
2  FORMAT(A3)
C
C *** CHECK ANSWER ENTRY ***
C
CALL ANSCHK(ANSWER)
C
IF(ANSWER.EQ.'NO')GO TO 3
C
C *** CHECK ENTRY INFORMATION ***
C
CALL IRCCHK
C
3  RETURN
   END

```

```

SUBROUTINE IRCCHK
C
C THIS SUBROUTINE ALLOWS THE USER TO CHECK THE INFORMATION
C INPUT FOR THE NUMBER OF ROWS AND COLUMNS OF THE IMAGE, AND
C TO CORRECT THEM IF DESIRED.
C
COMMON /IMAGE/ IMGROW,IMGCOL
1 TYPE *,' '
TYPE *,'HERE IS WHAT YOU HAVE INPUT:'
TYPE *,' '
C
C *** DISPLAY THE NUMBER OF IMAGE ROWS AND COLUMNS ***
C
TYPE *,'NUMBER OF IMAGE ROWS = ',IMGROW
TYPE *,'NUMBER OF IMAGE COLUMNS = ',IMGCOL
C
C *** VERIFY THE CORRECT RESPONSE ***
C
CALL VERIFY(ANSWER)
C
IF(ANSWER.EQ.'YES')GO TO 2
C
C *** CHANGE INCORRECT INFORMATION ***
C
CALL IRCCOR
C
GO TO 1
2 RETURN
END

```

```

SUBROUTINE IRCCOR
C
C THIS SUBROUTINE ALLOWS THE USER TO CORRECT THE INFORMATION
C INPUT FOR THE NUMBER OF IMAGE ROWS AND COLUMNS.
C
REAL ITEM
C
C *** MAKE ERROR CORRECTIONS ***
C
TYPE *, ' '
TYPE *, 'DO YOU WANT TO CHANGE THE NUMBER OF ROWS, COLUMNS,'
TYPE *, 'OR BOTH?'
1 TYPE *, 'IF YOU WANT TO CHANGE THE NUMBER OF ROWS, TYPE ROWS.'
TYPE *, 'IF YOU WANT TO CHANGE THE NUMBER OF COLUMNS, TYPE COLS.'
TYPE *, 'IF YOU WANT TO CHANGE BOTH, TYPE BOTH.'
TYPE *, ' '
TYPE 2
2 FORMAT(5X, 'ITEM YOU WISH TO CHANGE = ', $)
ACCEPT 3, ITEM
3 FORMAT(A4)
IF(ITEM.EQ.'BOTH')GO TO 10
IF(ITEM.EQ.'ROWS')GO TO 20
IF(ITEM.EQ.'COLS')GO TO 30
TYPE *, ' '
TYPE *, '*** ITEM ENTRY ERROR - PLEASE RETYPE ***'
TYPE *, ' '
GO TO 1
C
C *** CHANGE BOTH NUMBER OF ROWS AND COLUMNS ***
C
10 CALL IMROW
CALL IMCOL
RETURN
C
C *** CHANGE NUMBER OF ROWS ***
C
20 CALL IMROW
RETURN
C
C *** CHANGE NUMBER OF COLUMNS ***
C
30 CALL IMCOL
RETURN
END

```

```

SUBROUTINE IMROW

C      THIS SUBROUTINE ALLOWS THE USER TO INPUT THE NUMBER OF ROWS OF
C      THE IMAGE TO BE INPUT OR OUTPUT.
C
C      INCLUDE 'COMMON.FTN'
C      COMMON /IMAGE/ IMGROW
C
C      *** ENTER THE NUMBER OF IMAGE ROWS ***
C
1      TYPE 2
2      FORMAT(5X,'NUMBER OF IMAGE ROWS = ',S)
      ACCEPT *,IMGROW
C
      IF(IMGROW.LT.5)GO TO 10
      IF(IMGROW.GT.M)GO TO 20
      RETURN
10     TYPE *,'
      TYPE *,'*** IMAGE ROW SIZE EXCESSIVELY SMALL ***'
      TYPE *,'
      TYPE *,'      IMAGE ROW SIZE INPUT = ',IMGROW
      TYPE *,'
      TYPE *,'DO YOU WISH TO CHANGE THE IMAGE ROW SIZE INPUT?'
      TYPE *,'IF YOU DO, TYPE YES.  OTHERWISE, TYPE NO.'
      TYPE *,'
      TYPE 11
11     FORMAT (5X,'ANSWER = ',S)
      ACCEPT 12,ANSWER
12     FORMAT(A3)
      CALL ANSCHK(ANSWER)
      IF(ANSWER.EQ.'YES')GO TO 1
      RETURN
20     TYPE *,'
      TYPE *,'IMAGE ROW SIZE EXCEEDS MAXIMUM ALLOWABLE COLUMN SIZE:',M
      TYPE *,'
      TYPE *,'ARE YOU SURE YOU WANT TO USE THIS IMAGE ROW SIZE?'
      TYPE *,'IF YOU ARE, TYPE YES.  OTHERWISE, TYPE NO.'
      TYPE *,'
      TYPE 11
      ACCEPT 12,ANSWER
      CALL ANSCHK(ANSWER)
      IF(ANSWER.EQ.'NO')GO TO 1
      RETURN
      END

```

```

SUBROUTINE IMCOL
C
C THIS SUBROUTINE ALLOWS THE USER TO INPUT THE NUMBER OF COLUMNS
C OF THE IMAGE TO BE INPUT OR OUTPUT.
C
INCLUDE 'COMMON.FTN'
COMMON /IMAGE/ IX,IMGCOL
C
C *** ENTER THE NUMBER OF IMAGE COLUMNS ***
C
TYPE 1
1  FORMAT(5X,'NUMBER OF IMAGE COLUMNS = ',S)
   ACCEPT *,IMGCOL
   IF(IMGCOL.GT.M)GO TO 2
   IF(IMGCOL.LT.5)GO TO 3
C
RETURN
2  TYPE *,' '
   TYPE *,'*** MAXIMUM IMAGE COLUMN SIZE',M,' EXCEEDED IN IMCOL ***'
   GO TO 4
3  TYPE *,' '
   TYPE *,'*** IMAGE COLUMN SIZE EXCESSIVELY SMALL ***'
C
C *** CORRECT COLUMN SIZE ***
C
4  CALL IRCCHK
C
   IF(IMGCOL.GT.M)GO TO 2
   RETURN
END

```



```

SUBROUTINE VERIFY(ANSWER)
C
C THIS SUBROUTINE ALLOWS THE USER TO VERIFY THE RESPONSE
C GIVEN TO THE INFORMATION INPUT.
C
TYPE *, ' '
TYPE *, 'IS THIS THE CORRECT INFORMATION INPUT?'
TYPE *, 'IF SO, TYPE YES. OTHERWISE, NO.'
TYPE *, ' '
TYPE 1
1 FORMAT(5X, 'ANSWER = ', $)
ACCEPT 2, ANSWER
2 FORMAT(A3)
C
C *** CHECK ANSWER RESPONSE ***
C
CALL ANSCHK(ANSWER)
C
RETURN
END

SUBROUTINE ANSCHK(ANSWER)
C
C THIS SUBROUTINE ALLOWS THE USER TO CHECK A YES/NO ANSWER
C RESPONSE FOR AN INCORRECT ENTRY.
C
IF(ANSWER.EQ.'YES'.OR.ANSWER.EQ.'NO')RETURN
C
C *** CORRECT ANSWER RESPONSE ***
C
CALL ANSCOR(ANSWER)
C
RETURN
END

SUBROUTINE ANSCOR(ANSWER)
C
C THIS SUBROUTINE ALLOWS THE USER TO CORRECT AN INCORRECT
C YES/NO ANSWER ENTRY.
C
1 TYPE *, ' '
TYPE *, '*** ANSWER ENTRY ERROR - PLEASE RETYPE ***'
TYPE *, ' '
TYPE 2
2 FORMAT(5X, 'ANSWER = ', $)
ACCEPT 3, ANSWER
3 FORMAT(A3)

```

[illegible]

```

SUBROUTINE IMOSL(IMOERR,IMOEND)

C
C THIS SUBROUTINE ALLOWS THE USER TO SELECT THE IMAGE OPERATION
C DESIRED. THE IMAGE OPERATION IS THEN PERFORMED.
C
1  TYPE *, '
TYPE *, 'ENTER THE NUMBER CORRESPONDING TO THE IMAGE OPERATION'
TYPE *, 'YOU WISH TO PERFORM:'
TYPE *, '
TYPE *, ' 1. PRINT IMAGE INTENSITIES'
TYPE *, ' 2. DELETE AN IMAGE FILE'
TYPE *, ' 3. CREATE A NEW IMAGE FILE'
TYPE *, ' 4. NONE YET'
TYPE *, ' 5. NONE YET'
TYPE *, ' 6. TERMINATE IMAGE OPERATIONS'
TYPE *, ' 7. TERMINATE PROGRAM EXECUTION'
TYPE *, '
TYPE 5
5  FORMAT(5X,'OPERATION NUMBER = ',S)
ACCEPT *,IMONBR
IF(IMONBR.LT.1.OR.IMONBR.GT.7)GO TO 100
GO TO(10,20,30,40,50,60,70)IMONBR
10 CALL IMOPR1(IMOERR)
RETURN
20 CALL IMOPR2(IMOERR)
RETURN
30 CALL IMOPR3(IMOERR,IMOEND)
RETURN
40 CALL IMOPR4(IMOERR)
GO TO 1
50 CALL IMOPR5(IMOERR)
GO TO 1
60 RETURN
70 IMOEND = 1
RETURN
100 TYPE *, '
TYPE *, '*** INVALID OPERATION NUMBER',IMONBR,' ATTEMPTED ***'
GO TO 1
END

```

```

SUBROUTINE IMOPRI(IMOERR)
C
C THIS SUBROUTINE ALLOWS THE USER TO PRINT AN IMAGE FILE ON THE
C LINE PRINTER. THE LOGICAL UNIT NUMBER AND FILE NAME ARE ENTERED
C INTERACTIVELY, AND THE USER MAY CORRECT THEM IF AN ENTRY ERROR
C OCCURS.
C
C INCLUDE 'COMMON.FTN'
COMMON /BUF/ IOBUFR(L,M)
COMMON /INBUF/ INTBUF(M)
COMMON /IMAGE/ IMGROW,IMGCOL
LOGICAL*1 IOBUFR
INTEGER RECNUM
D WRITE (6,*) '*** ENTER IMOPRI ***'
C
C *** ENTER THE LOGICAL UNIT NUMBER AND FILE NAME ***
C
C TYPE *, ' '
C TYPE *, 'PLEASE ENTER THE LOGICAL UNIT NUMBER AND FILE NAME FOR'
C TYPE *, 'THE IMAGE FILE YOU WISH TO PRINT:'
CALL FLUFND
C
C *** OPEN THE FILE DEFINED BY THE LOGICAL UNIT NUMBER AND FILENAME
C PREVIOUSLY ENTERED ***
C
CALL OPNOLD(IMOERR)
IF(IMOERR.EQ.1)RETURN
C
DO 30 RECNUM=1,IMGROW
C
C *** READ THE IMAGE INTENSITY DATA FROM THE IMAGE FILE ***
C
I = 1
CALL INPUT(RECNUM,I,IMOERR)
IF(IMOERR.EQ.1)RETURN
C
C *** PERFORM BYTE TO WORD NUMBER CONVERSION ***
C
DO 10 J=1,IMGCOL
INTBUF(J) = IOBUFR(1,J)
IF(INTBUF(J).LT.0)INTBUF(J) = INTBUF(J) + 256
10 CONTINUE
C
C *** WRITE THE IMAGE INTENSITY DATA TO THE LINE PRINTER ***
C
WRITE (6,*) 'RECORD NUMBER',RECNUM,' : '
WRITE (6,15)
15 FORMAT(1X,24('-'))
WRITE (6,20)(INTBUF(J),J=1,IMGCOL)

```

```
20  FORMAT(26(2X,I3),/)  
    WRITE (6,*) ' '  
C  
30  CONTINUE  
C  
C    *** CLOSE THE IMAGE FILE ***  
C    CALL CLOSKP(IMOERR)  
C  
    RETURN  
    END
```

SUBROUTINE IMOPR2(IMOERR)

THIS SUBROUTINE ALLOWS THE USER TO DELETE A SPECIFIED FILE.
THE LOGICAL UNIT NUMBER AND FILE NAME ARE ENTERED INTER-
ACTIVELY, AND THE USER MAY CORRECT THEM IF AN ENTRY ERROR
OCCURS.

WRITE (6,*) '*** ENTER IMOPR2 ***'

*** ENTER THE LOGICAL UNIT NUMBER AND FILE NAME ***

TYPE *, ' '

TYPE *, 'PLEASE ENTER THE LOGICAL UNIT NUMBER AND FILE NAME FOR'

TYPE *, 'THE IMAGE FILE YOU WISH TO DELETE:'

CALL FLUFND

*** OPEN THE FILE TO BE DELETED WHICH IS DEFINED BY THE LOGICAL
UNIT NUMBER AND FILENAME PREVIOUSLY ENTERED ***

CALL OPNOLD(IMOERR)

*** CLOSE THE FILE WITH THE DELETE DISPOSITION OPTION ***

CALL CLOSDL(IMOERR)

IF(IMOERR.EQ.1)RETURN

TYPE *, ' '

TYPE *, '*** IMAGE FILE DELETED ***'

WRITE (6,*) ' '

WRITE (6,*) '*** IMAGE FILE DELETED ***'

WRITE (6,*) ' '

RETURN

END

```

SUBROUTINE IMOPR3(IMOERR,IMOEND)
C
C THIS SUBROUTINE CREATES A NEW IMAGE FILE FROM AN ALGORITHM
C WHICH THE USER SELECTS. THE LOGICAL UNIT NUMBER AND FILE
C NAME ARE ENTERED INTERACTIVELY, AND THE USER MAY CORRECT THEM
C IF AN ENTRY ERROR OCCURS.
C
C WRITE (6,*) '*** ENTER IMOPR3 ***'
C
C *** ENTER THE LOGICAL UNIT NUMBER AND FILE NAME ***
C
C TYPE *, ' '
C TYPE *, 'PLEASE ENTER THE LOGICAL UNIT NUMBER AND FILE NAME FOR '
C TYPE *, 'THE NEW IMAGE FILE YOU WISH TO CREATE:'
C CALL FLUFND
C
C *** OPEN THE FILE DEFINED BY THE LOGICAL UNIT NUMBER AND FILENAME
C PREVIOUSLY ENTERED ***
C
C CALL OPNNEW(IMOERR)
C IF(IMOERR.EQ.1)RETURN
C
C *** SELECT THE ALGORITHM TO BE USED TO CREATE THE NEW IMAGE ***
C
C CALL IMASEL(IMOERR,IMOEND)
C IF(IMOERR.EQ.1)RETURN
C
C *** CLOSE THE FILE PREVIOUSLY OPENED ***
C
C CALL CLOSKP(IMOERR)
C
C RETURN
C END

SUBROUTINE IMOPR4(IMOERR)
C
C TYPE *, '*** IMOPR4 NOT AVAILABLE AT THIS TIME ***'
C RETURN
C END

SUBROUTINE IMOPR5(IMOERR)
C
C TYPE *, '*** IMOPR5 NOT AVAILABLE AT THIS TIME ***'
C RETURN
C END

```

```

SUBROUTINE FLUFND
C
C THIS SUBROUTINE ALLOWS THE USER TO ENTER THE LOGICAL UNIT
C NUMBER AND FILE NAME INTERACTIVELY FOR IMAGE INPUT/OUTPUT
C OPERATIONS. THE USER MAY CORRECT THIS INFORMATION IF
C AN ENTRY ERROR OCCURS.
C
COMMON /FILE/ LUN,NCHAR,FNAME(34)
LOGICAL*1 FNAME
C
C *** ENTER THE LOGICAL UNIT NUMBER AND FILE NAME ***
C
CALL LUNBR
CALL FILENM
C
C *** OPTIONAL ENTRY INFORMATION CHECK ***
C
TYPE *, ' '
TYPE *, 'DO YOU WISH TO CHECK FOR ENTRY ERRORS?'
TYPE *, 'IF YOU DO, TYPE YES. OTHERWISE, TYPE NO.'
TYPE *, ' '
TYPE 1
1 FORMAT(5X, 'ANSWER = ', $)
ACCEPT 2, ANSWER
2 FORMAT(A3)
C
C *** CHECK ANSWER ENTRY ***
C
CALL ANSCHK(ANSWER)
C
IF(ANSWER.EQ.'NO')GO TO 3
C
C *** CHECK ENTRY INFORMATION ***
C
CALL FLUCHK
C
3 WRITE (6,*) ' '
WRITE (6,*) '*** IMAGE FILE INFORMATION ***'
WRITE (6,*) ' '
WRITE (6,*) 'LOGICAL UNIT NUMBER = ', LUN
WRITE (6,4) (FNAME(I), I=1, NCHAR)
4 FORMAT(1X, 'FILENAME = ', <NCHAR+1>A1)
WRITE (6,*) ' '
RETURN
END

```



```

SUBROUTINE FLUCHK
C
C THIS SUBROUTINE ALLOWS THE USER TO CHECK THE INFORMATION
C INPUT FOR THE FILE NAME AND LOGICAL UNIT NUMBER, AND TO
C CORRECT THEM IF DESIRED.
C
COMMON /FILE/ LUN,NCHAR,FNAME(34)
LOGICAL*1 FNAME
1 TYPE *, ' '
TYPE *, 'HERE IS WHAT YOU HAVE INPUT:'
TYPE *, ' '
C
C *** DISPLAY THE FILE NAME AND LOGICAL UNIT NUMBER INPUT ***
C
TYPE *, 'LOGICAL UNIT NUMBER = ', LUN
TYPE 2, (FNAME(I), I=1, NCHAR)
2 FORMAT(1X, 'FILENAME = ', <NCHAR+1>A1)
C
C *** VERIFY THE CORRECT RESPONSE ***
C
CALL VERIFY(ANSWER)
C
IF(ANSWER.EQ.'YES')GO TO 3
C
C *** CHANGE INCORRECT INFORMATION ***
C
CALL FLUCOR
C
GO TO 1
3 RETURN
END

```

```

SUBROUTINE FLUCOR
C
C THIS SUBROUTINE ALLOWS THE USER TO CORRECT THE INFORMATION
C INPUT FOR THE FILE NAME AND LOGICAL UNIT NUMBER.
C
REAL ITEM
C
C *** MAKE ERROR CORRECTIONS ***
C
TYPE *, ' '
TYPE *, 'DO YOU WANT TO CHANGE THE LOGICAL UNIT NUMBER, '
TYPE *, 'FILE NAME, OR BOTH?'
1 TYPE *, 'IF YOU WANT TO CHANGE THE LOGICAL UNIT NUMBER, TYPE LUN.'
TYPE *, 'IF YOU WANT TO CHANGE THE FILE NAME, TYPE FILE.'
TYPE *, 'IF YOU WANT TO CHANGE BOTH, TYPE BOTH.'
TYPE *, ' '
TYPE 2
2 FORMAT(5X, 'ITEM YOU WISH TO CHANGE = ', $)
ACCEPT 3, ITEM
3 FORMAT(A4)
IF(ITEM.EQ.'BOTH')GO TO 10
IF(ITEM.EQ.'LUN')GO TO 20
IF(ITEM.EQ.'FILE')GO TO 30
TYPE *, ' '
TYPE *, '*** ITEM ENTRY ERROR - PLEASE RETYPE ***'
TYPE *, ' '
GO TO 1
C
C *** CHANGE BOTH FILE NAME AND LOGICAL UNIT NUMBER ***
C
10 CALL FILENM
CALL LUNBR
RETURN
C
C *** CHANGE LOGICAL UNIT NUMBER ***
C
20 CALL LUNBR
RETURN
C
C *** CHANGE FILE NAME ***
C
30 CALL FILENM
RETURN
END

```

SUBROUTINE FILENM

C
C THIS SUBROUTINE ALLOWS THE USER TO INPUT THE FILE NAME OF
C THE FILE BEING USED FOR INPUT/OUTPUT OPERATIONS.
C

COMMON /FILE/ IX,NCHAR,FNAME(34)

LOGICAL*1 FNAME

TYPE *,'

TYPE *,'WHAT DEVICE/UIC/FILENAME DO YOU WISH TO USE?'

TYPE *,'ENTER ACCORDING TO THE FOLLOWING FORMAT:'

TYPE *,'

TYPE *,' DEV:[UIC]FILENAME.TYPE:VERSION'

TYPE *,'EXAMPLE: DK1:[10,20]FILENAME.IMG;3'

TYPE *,'

C
C *** INPUT THE DESIRED FILE NAME ***
C

TYPE 1

1 FORMAT(5X,'FILENAME = ',S)

ACCEPT 2,NCHAR,FNAME

2 FORMAT(Q,34A1)

FNAME(NCHAR+1) = 0

C
RETURN
END

```

SUBROUTINE LUNBR
C
C THIS SUBROUTINE ALLOWS THE USER TO INPUT THE LOGICAL UNIT
C NUMBER ASSOCIATED WITH THE FILE TO BE USED FOR INPUT AND
C OUTPUT OPERATIONS.
C
COMMON /FILE/ LUN
1 TYPE *, ' '
TYPE *, 'WHAT LOGICAL UNIT NUMBER YOU WISH TO USE (1-10)?'
TYPE *, ' '
C
C *** INPUT THE DESIRED LOGICAL UNIT NUMBER ***
C
TYPE 2
2 FORMAT(5X, 'LOGICAL UNIT NUMBER = ', S)
ACCEPT *, LUN
C
C *** CHECK FOR INCORRECT NUMBER ***
C
IF(LUN.LT.1.OR.LUN.GT.10)GO TO 3
IF(LUN.EQ.5)GO TO 4
IF(LUN.EQ.6)GO TO 5
C
RETURN
3 TYPE *, ' '
TYPE *, '*** INVALID LOGICAL UNIT NUMBER', LUN, ' ATTEMPTED ***'
TYPE *, ' '
TYPE *, 'LOGICAL UNIT NUMBER MUST BE BETWEEN 1 AND 10.'
GO TO 1
4 TYPE *, ' '
TYPE *, 'LOGICAL UNIT 5 IS RESERVED FOR THE USERS TERMINAL.'
TYPE *, 'PLEASE USE ANOTHER NUMBER!'
GO TO 1
5 TYPE *, ' '
TYPE *, 'LOGICAL UNIT 6 IS RESERVED FOR THE LINE PRINTER.'
TYPE *, 'PLEASE USE ANOTHER NUMBER!'
GO TO 1
END

```

```

SUBROUTINE IMASEL(IMAERR, IMAEND)

C
C THIS SUBROUTINE ALLOWS THE USER TO SELECT THE ALGORITHM
C TO BE USED TO FIND THE INTENSITY VALUES OF THE IMAGE
C ELEMENTS OF THE NEW IMAGE TO BE CREATED.
C
1  TYPE *, '
TYPE *, 'ENTER THE NUMBER CORRESPONDING TO THE ALGORITHM YOU WISH'
TYPE *, 'TO USE TO FIND THE INTENSITIES OF THE IMAGE ELEMENTS:'
TYPE *, '
TYPE *, '    1. RANDOM INTENSITY DISTRIBUTION'
TYPE *, '    2. USER TERMINAL INPUT'
TYPE *, '    3. CARD READER INPUT'
TYPE *, '    4. NONE YET'
TYPE *, '    5. NONE YET'
TYPE *, '    6. TERMINATE PROGRAM EXECUTION'
TYPE *, '
TYPE 5
5  FORMAT(5X, 'ALGORITHM NUMBER = ', S)
ACCEPT *, IMANBR
IF(IMANBR.LT.1.OR.IMANBR.GT.6)GO TO 100
GO TO(10,20,30,40,50,60)IMANBR
10 CALL IMALG1(IMAERR)
RETURN
20 CALL IMALG2(IMAERR)
RETURN
30 CALL IMALG3(IMAERR)
RETURN
40 CALL IMALG4(IMAERR)
GO TO 1
50 CALL IMALG5(IMAERR)
GO TO 1
60 IMAEND = 1
RETURN
100 TYPE *, '
TYPE *, '*** INVALID ALGORITHM NUMBER', IMANBR, ' ATTEMPTED ***'
GO TO 1
END

```

```

SUBROUTINE IMALGI(IMAERR)
C
C THIS SUBROUTINE ASSIGNS RANDOMLY SELECTED VALUES TO THE
C IMAGE ELEMENTS OF THE IMAGE BEING CREATED, THEN OUTPUTS THEM.
C
INCLUDE 'COMMON.FTN'
COMMON /BUF/ IOBUFR(L,M)
COMMON /INBUF/ INTBUF(M)
COMMON /IMAGE/ IMGROW,IMGCOL
LOGICAL*1 IOBUFR
INTEGER RECNUM
INTEGER*4 SEED
D WRITE (6,*) '*** ENTER IMALGI ***'
C
C *** ASSIGN SEED AN INITIAL VALUE ***
C
CALL SEEDFD(SEED)
C
DO 20 RECNUM=1,IMGROW
DO 10 J=1,IMGCOL
C
C *** ASSIGN THE INTENSITY VALUES ***
C
INTBUF(J) = RAN(SEED)*256.0
IOBUFR(1,J) = INTBUF(J)
C
10 CONTINUE
C
C *** WRITE TO THE LINE PRINTER ***
C
D WRITE (6,*) 'RECORD NUMBER',RECNUM,' : '
D WRITE (6,14)
D14 FORMAT(1X,24('-'))
D WRITE (6,15)(INTBUF(J),J=1,IMGCOL)
D15 FORMAT(26(2X,13))
D WRITE (6,*) ' '
C
C *** OUTPUT EACH RECORD TO THE IMAGE FILE ***
C
I = 1
CALL OUTPUT(RECNUM,I,IMAERR)
IF(IMAERR.EQ.1)RETURN
C
20 CONTINUE
RETURN
END

```

```

SUBROUTINE SEEDFD(SEED)
C
C THIS SUBROUTINE ALLOWS THE USER TO INTERACTIVELY INPUT A
C SEED VALUE TO BE USED BY THE RANDOM NUMBER GENERATOR FOR
C CALCULATING RANDOM INTENSITY VALUES.
C
INTEGER*4 SEED
C
C *** ASSIGN SEED AN INITIAL VALUE ***
C
ANSWER = 'YES'
TYPE *, ' '
TYPE *, 'WHAT IS THE SEED VALUE YOU WISH TO USE FOR THE'
TYPE *, 'RANDOM NUMBER GENERATOR?'
TYPE *, 'EACH DIFFERENT SEED VALUE GENERATES A DIFFERENT'
TYPE *, 'RANDOM SEQUENCE OF NUMBERS FROM 0 TO 255.'
1 TYPE *, ' '
IF(ANSWER.EQ.'NO')TYPE *, 'INPUT A NEW SEED VALUE'
TYPE *, ' '
TYPE 2
2 FORMAT(5X, 'SEED = ', SEED)
ACCEPT *, SEED
TYPE *, ' '
TYPE *, 'HERE IS WHAT YOU HAVE INPUT:'
TYPE *, ' '
C
C *** DISPLAY THE SEED VALUE ***
C
TYPE *, 'SEED = ', SEED
TYPE *, ' '
C
C *** VERIFY THE CORRECT RESPONSE ***
C
CALL VERIFY(ANSWER)
C
IF(ANSWER.EQ.'NO')GO TO 1
WRITE (6,*) ' '
WRITE (6,*) '*** SEED VALUE FOR RANDOM NUMBER GENERATOR ***'
WRITE (6,*) ' '
WRITE (6,*) 'SEED = ', SEED
WRITE (6,*) ' '
RETURN
END

```

```

SUBROUTINE IMALG2(IMAERR)

C
C THIS SUBROUTINE ASSIGNS INTENSITY VALUES TO THE IMAGE
C ELEMENTS OF THE IMAGE BEING CREATED USING THE USER'S
C TERMINAL. THE INTENSITY VALUES ARE ENTERED ONE RECORD
C AT A TIME. EACH VALUE IS SEPARATED BY ONE OR MORE SPACES
C OR TABS, OR A COMMA (WITH OR WITHOUT SURROUNDING SPACES).
C EACH RECORD IS TERMINATED WITH A SLASH (/). INTENSITY
C DATA MAY BE LISTED AS A SINGLE VALUE, A REPETITION OF
C THE FORM (NUM-OF-REPS*VALUE), OR A REPETITION OF ZEROS
C OF THE FORM (NUM-OF-REPS* ), WHERE NUM-OF-REPS REPRESENTS
C THE NUMBER CORRESPONDING TO THE NUMBER OF TIMES THE VALUE
C IS TO BE SUCCESSIVELY REPEATED.

C
C INCLUDE 'COMMON.FTN'
C COMMON /BUF/ IOBUFR(L,M)
C COMMON /INBUF/ INTBUF(M)
C COMMON /IMAGE/ IMGROW,IMGCOL
C LOGICAL*1 IOBUFR
C INTEGER RECNUM
D WRITE (6,*) '*** ENTER IMALG2 ***'
C TYPE *, ' '
C TYPE *, 'ENTER INTENSITY VALUES ONE RECORD AT A TIME.'
C TYPE *, 'SEPARATE EACH VALUE WITH ONE OR MORE SPACES OR TABS,'
C TYPE *, 'OR A COMMA (WITH OR WITHOUT SURROUNDING SPACES).'
C TYPE *, 'TERMINATE EACH RECORD WITH A SLASH (/).'
C TYPE *, 'VALUES MAY BE REPEATED USING THE FORM (NUM-OF-REPS*VALUE)'
C TYPE *, 'OR (NUM-OF-REPS* ) FOR ZEROS.'
C TYPE *, ' '
C TYPE *, 'EXAMPLE:'
C TYPE *, ' '
C TYPE *, 'ENTER RECORD NUMBER      5 [END WITH / AND <CR>]'
C TYPE *, ' '
C TYPE *, ' 3, 7 12*37.245, 16*, 2,0, 100 /'
C TYPE *, ' '
C TYPE *, 'IN THE ABOVE EXAMPLE, THE FIRST TWO RECORD VALUES WILL'
C TYPE *, 'BE 3 AND 7, FOLLOWED BY TWELVE VALUES OF 37.245, SIX-'
C TYPE *, 'TEEN VALUES OF ZERO, AND THE NUMBERS 2, ZERO, AND 100.'
C TYPE *, ' '
C DO 20 RECNUM=1,IMGROW
C
C *** ENTER EACH RECORD INTERACTIVELY ***
C
C TYPE *, 'ENTER RECORD NUMBER',RECNUM,' [END WITH / AND <CR>]'
C TYPE *, ' '
C ACCEPT *,(IOBUFR(1,J),J=1,IMGCOL)
C
C *** PERFORM BYTE TO WORD NUMBER CONVERSION ***

```



```

D      DO 5 J=1,IMGCOL
D      INTBUF(J) = IOBUFR(1,J)
D      IF(INTBUF(J).LT.0)INTBUF(J) = INTBUF(J) + 256
D5     CONTINUE
C
C      *** WRITE TO LINE PRINTER ***
C
D      WRITE (6,*) 'RECORD NUMBER',RECNUM,' :'
D      WRITE (6,9)
D9     FORMAT(1X,24('-'))
D      WRITE (6,10)(INTBUF(J),J=1,IMGCOL)
D10    FORMAT(26(2X,I3))
D      WRITE (6,*) ' '
C
C      *** OUTPUT EACH RECORD TO THE IMAGE FILE ***
C
      I=1
      CALL OUTPUT(RECNUM,I,IMAERR)
      IF(IMAERR.EQ.1)RETURN
C
20     CONTINUE
      RETURN
      END

```

```

SUBROUTINE IMALG3(IMAERR)
C
C THIS SUBROUTINE ASSIGNS INTENSITY VALUES TO THE IMAGE
C ELEMENTS OF THE IMAGE BEING CREATED USING THE CARD READER.
C EACH CARD CONTAINS A SERIES OF INTENSITY VALUES SEPARATED
C BY ONE OR MORE SPACES OR TABS, OR A COMMA (WITH OR WITHOUT
C SURROUNDING SPACES). EACH RECORD (NOT CARD) IS TERMINATED
C WITH A SLASH (/). INTENSITY VALUES MAY BE LISTED AS A SINGLE
C NUMBER, A REPETITION OF THE FORM (NUM-OF-REPS*VALUE), OR A
C REPETITION OF ZEROS OF THE FORM (NUM-OF-REPS* ), WHERE NUM-
C OF-REPS REPRESENTS THE NUMBER CORRESPONDING TO THE NUMBER OF
C TIMES THE VALUE IS TO BE SUCCESSIVELY REPEATED.
C
C INCLUDE 'COMMON.FTN'
C COMMON /BUF/ IOBUFR(L,M)
C COMMON /INBUF/ INTBUF(M)
C COMMON /IMAGE/ IMGROW,IMGCOL
C COMMON /FILE/ LUN
C LOGICAL*1 IOBUFR
C INTEGER RECNUM
D WRITE (6,*) '*** ENTER IMALG3 ***'
LUNCR = LUN + 1
IF(LUNCR.EQ.5)LUNCR = 7
IF(LUNCR.GT.10)LUNCR = 1
CALL ASSIGN(LUNCR,'CR:')
TYPE *, ' '
TYPE *, '*** PLEASE CHECK TO SEE IF THE CARD READER IS READY ***'
TYPE *, ' '
TYPE 1
1 FORMAT(5X,'IF READY, TYPE GO:',$,)
ACCEPT 2,GO
2 FORMAT(A2)
DO 20 RECNUM=1,IMGROW
C
C *** INPUT THE IMAGE DATA FROM THE CARD READER ***
C
C READ (LUNCR,*,END=20,ERR=100)(IOBUFR(1,J),J=1,IMGCOL)
C
C *** PERFORM BYTE TO WORD NUMBER CONVERSION ***
C
C DO 5 J=1,IMGCOL
D INTBUF(J) = IOBUFR(1,J)
D IF(INTBUF(J).LT.0)INTBUF(J) = INTBUF(J) + 256
D5 CONTINUE
C
C *** WRITE TO LINE PRINTER ***
C
D WRITE (6,*) 'RECORD NUMBER',RECNUM,' :'
D WRITE (6,9)

```

```

D9      FORMAT(1X,24('-'))
D       WRITE (6,10)(INTBUF(J),J=1,IMGCOL)
D10     FORMAT(26(2X,I3))
D       WRITE (6,*) ' '
C
C       *** OUTPUT EACH RECORD TO THE IMAGE FILE ***
C
      I = 1
      CALL OUTPUT(RECNUM,I,IMAERR)
      IF(IMAERR.EQ.1)RETURN
C
20      CONTINUE
      RETURN
100     TYPE *, ' '
      TYPE *, '*** READ ERROR OCCURED IN IMALG3 ON LOGICAL UNIT',LUNCR,
      1' ***'
      CALL CLOSDL(IMAERR)
      IMAERR = 1
      RETURN
      END

      SUBROUTINE IMALG4(IMAERR)
      TYPE *, '*** IMALG4 NOT AVAILABLE AT THIS TIME ***'
      TYPE *, ' '
      TYPE *, '*** [WARNING]: NO IMAGE WAS CREATED ***'
      TYPE *, ' '
      RETURN
      END

      SUBROUTINE IMALG5(IMAERR)
      TYPE *, '*** IMALG5 NOT AVAILABLE AT THIS TIME ***'
      TYPE *, ' '
      TYPE *, '*** [WARNING]: NO IMAGE WAS CREATED ***'
      TYPE *, ' '
      RETURN
      END

```

```

SUBROUTINE INPUT(RECNUM,I,INPERR)
C
C THIS SUBROUTINE INPUTS A ROW OF INTENSITY DATA INTO THE I/O
C BUFFER ONE RECORD LONG.
C
INCLUDE 'COMMON.FTN'
COMMON /BUF/ IOBUFR(L,M)
COMMON /IMAGE/ IX,IMGCOL
COMMON /FILE/ LUN
LOGICAL*1 IOBUFR
INTEGER RECNUM
C
C *** READ THE IMAGE INTENSITY DATA FROM THE IMAGE FILE ***
C
READ (LUN,RECNUM,END=10,ERR=100)(IOBUFR(I,J),J=1,IMGCOL)
C
10 CONTINUE
RETURN
100 TYPE *, ' '
TYPE *, '*** READ ERROR OCCURED IN INPUT ON LOGICAL UNIT',LUN,
1' ***'
CALL CLOSKP(INPERR)
INPERR = 1
WRITE (6,*) '*** EXIT INPUT - READ ERROR TERMINATION ***'
RETURN
END

```

```

SUBROUTINE OUTPUT(RECNUM,I,OUTERR)
C
C   THIS SUBROUTINE OUTPUTS A ROW OF INTENSITY DATA ONE RECORD
C   LONG TO THE OUTPUT FILE SPECIFIED.
C
  INCLUDE 'COMMON.FTN'
  COMMON /BUF/ IOBUFR(L,M)
  COMMON /IMAGE/ IX,IMGCOL
  COMMON /FILE/ LUN
  LOGICAL*1 IOBUFR
  INTEGER RECNUM,OUTERR

C
C   *** WRITE INTENSITY VALUES TO THE IMAGE FILE ***
C
  WRITE (LUN,RECNUM,ERR=100)(IOBUFR(I,J),J=1,IMGCOL)
C
  RETURN
100  TYPE *, ' '
  TYPE *, '*** WRITE ERROR OCCURED IN OUTPUT ON LOGICAL UNIT',LUN,
  1' ***'
  CALL CLOSDL(OUTERR)
  OUTERR = 1
  WRITE (6,*) '*** EXIT OUTPUT - WRITE ERROR TERMINATION ***'
  RETURN
END

```

```

SUBROUTINE OPNNEW(OPNERR)
C
C THIS SUBROUTINE OPENS A NEW FILE DEFINED BY THE LOGICAL UNIT
C NUMBER AND FILENAME PREVIOUSLY ENTERED.
C
COMMON /FILE/ LUN,IX,FNAME(34)
COMMON /IMAGE/ IMGROW,IMGCOL
LOGICAL*1 FNAME
INTEGER OPNERR
RECCOL = IMGCOL/4.0
IMGREC = IMGCOL/4
REMNR = RECCOL - FLOAT(IMGREC)
IF(REMNR.NE.0.0)IMGREC = IMGREC + 1
C
C *** OPEN THE FILE DEFINED BY THE LOGICAL UNIT NUMBER AND FILENAME
C PREVIOUSLY ENTERED ***
C
OPEN(UNIT=LUN,NAME=FNAME,DISPOSE='KEEP',ACCESS='DIRECT',
1 ORGANIZATION='SEQUENTIAL',TYPE='UNKNOWN',RECORDSIZE=IMGREC,
2 MAXREC=IMGROW,BLOCKSIZE=IMGREC,INITIALSIZE=IMGROW,ERR=100,
3 FORM='UNFORMATTED')
C
C THE FILE OPENED ABOVE IS STRUCTURED AS AN UNFORMATTED DIRECT
C ACCESS SEQUENTIAL FILE. THE LENGTH OF EACH FILE RECORD IS
C DETERMINED BY THE NUMBER OF COLUMNS OF THE IMAGE TO BE OUTPUT.
C THE NUMBER OF RECORDS IN THE FILE IS DETERMINED BY THE NUMBER
C OF ROWS OF THE IMAGE TO BE OUTPUT. THESE PARAMETERS WERE INPUT
C IN SUBROUTINE IMSZ.
C
RETURN
100 TYPE *,' '
TYPE *,'*** OPEN ERROR OCCURED IN OPNNEW ON LOGICAL UNIT',LUN,
1 ' ***'
CALL CLOSDL(OPNERR)
OPNERR = 1
WRITE (6,*) '*** EXIT OPNNEW - OPEN ERROR TERMINATION ***'
RETURN
END

```

```

SUBROUTINE OPNOLD(OPNERR)
C
C THIS SUBROUTINE OPENS AN OLD FILE DEFINED BY THE LOGICAL UNIT
C NUMBER AND FILENAME PREVIOUSLY ENTERED.
C
COMMON /FILE/ LUN,IX,FNAME(34)
COMMON /IMAGE/ IMGROW,IMGCOL
LOGICAL*1 FNAME
INTEGER OPNERR
RECCOL = IMGCOL/4.0
IMGREC = IMGCOL/4
REMNR = RECCOL - FLOAT(IMGREC)
IF(REMNR.NE.0.0)IMGREC = IMGREC + 1
C
C *** OPEN THE FILE DEFINED BY THE LOGICAL UNIT NUMBER AND FILENAME
C PREVIOUSLY ENTERED ***
C
OPEN(UNIT=LUN,NAME=FNAME,DISPOSE='KEEP',ACCESS='DIRECT',
1 ORGANIZATION='SEQUENTIAL',TYPE='OLD',RECORDSIZE=IMGREC,
2 MAXREC=IMGROW,BLOCKSIZE=IMGREC,INITIALSIZE=IMGROW,ERR=100,
3 FORM='UNFORMATTED')
C
C THE FILE OPENED ABOVE IS STRUCTURED AS AN UNFORMATTED DIRECT
C ACCESS SEQUENTIAL FILE. THE LENGTH OF EACH FILE RECORD IS
C DETERMINED BY THE NUMBER OF COLUMNS OF THE IMAGE TO BE INPUT.
C THE NUMBER OF RECORDS IN THE FILE IS DETERMINED BY THE NUMBER
C OF ROWS OF THE IMAGE TO BE INPUT. THESE PARAMETERS WERE INPUT
C IN SUBROUTINE IMSIZE.
C
RETURN
100 TYPE *, ' '
TYPE *, '*** OPEN ERROR OCCURED IN OPNOLD ON LOGICAL UNIT',LUN,
1 ' ***'
CALL CLOSKP(OPNERR)
OPNERR = 1
WRITE (6,*) '*** EXIT OPNOLD - OPEN ERROR TERMINATION ***'
RETURN
END

```

```

SUBROUTINE CLOSKP(CLSERR)
C
C THIS SUBROUTINE CLOSES THE FILE PREVIOUSLY ASSIGNED TO LOGICAL
C UNIT NUMBER LUN, THEN KEEPS IT.
C
COMMON /FILE/ LUN
INTEGER CLSERR
C
C *** CLOSE THE FILE PREVIOUSLY OPENED ***
C
CLOSE(UNIT=LUN,DISPOSE='KEEP',ERR=100)
C
RETURN
100 TYPE *, '
TYPE *, '*** CLOSE ERROR OCCURED IN CLOSKP ON LOGICAL UNIT',LUN,
1' ***'
CLSERR = 1
WRITE (6,*) '*** EXIT CLOSKP - CLOSE ERROR TERMINATION ***'
RETURN
END

```



```

SUBROUTINE CLOSDL(CLSERR)
C
C THIS SUBROUTINE CLOSES THE FILE PREVIOUSLY ASSIGNED TO LOGICAL
C UNIT NUMBER LUN, THEN DELETES IT.
C
COMMON /FILE/ LUN
INTEGER CLSERR
C
C *** CLOSE THE FILE PREVIOUSLY OPENED ***
C
CLOSE(UNIT=LUN,DISPOSE='DELETE',ERR=100)
C
RETURN
100 TYPE *, ' '
TYPE *, '*** CLOSE ERROR OCCURED IN CLOSDL ON LOGICAL UNIT', LUN,
1 ' ***'
CLSERR = 1
WRITE (6,*) '*** EXIT CLOSDL - CLOSE ERROR TERMINATION ***'
RETURN
END

```

Appendix D: Interesting Point Selection Section

This appendix contains the FORTRAN source listings for the Interesting Point Selection Section of program DIDA.

```

SUBROUTINE INTPTS(INT,INTERR,INTEND)
C
C THIS SUBROUTINE IS USED TO OBTAIN THE INTERESTING POINTS OF
C THE SELECTED IMAGE. THE IMAGE SIZE IS DEFINED FIRST. NEXT,
C THE SIZE OF THE WINDOW USED FOR PROCESSING IS DEFINED. THE
C THE SELECTED IMAGE IS THEN INPUT. FINALLY, THE INTEREST OPER-
C ATOR TO BE USED TO FIND THE INTERESTING POINTS IS SELECTED
C AND PROCESSING BEGINS. WHEN COMPLETE, THE LOCATION OF THE
C INTERESTING POINTS ON THE SELECTED IMAGE AND THEIR CORRESPOND-
C ING VALUE, TYPE, AND CORRELATION WINDOW ARE STORED ON AN
C EXTERNAL FILE IF THE USER DESIRES. THESE POINTS ARE ALSO
C LISTED ON THE LINE PRINTER IF THE USER DESIRES.
C
COMMON /IOP/ IX,IPFLAG
D WRITE (6,*) '*** ENTER INTPTS ***'
IPFLAG = 0
C
C *** DEFINE THE IMAGE SIZE ***
C
CALL IMGSize
C
C *** DEFINE THE WINDOW SIZE ***
C
CALL WNDSize
C
C *** INPUT THE SELECTED IMAGE ***
C
CALL IMGin(INT,INTERR)
IF(INTERR.EQ.1)RETURN
C
C *** TEST FOR STORAGE ***
C
TYPE *, ' '
TYPE *, 'DO YOU WISH TO STORE THE INTERESTING POINTS ON AN'
TYPE *, 'OUTPUT FILE?'
TYPE *, 'IF YOU DO, TYPE YES. OTHERWISE, TYPE NO.'
TYPE *, ' '
TYPE 1
1 FORMAT(5X, 'ANSWER = ', $)
ACCEPT 2, ANSWER
2 FORMAT(A3)
CALL ANSCHK(ANSWER)
```

```

IF(ANSWER.EQ.'NO')GO TO 10
IPFLAG = 1
C
C
C
*** DEFINE THE CORRELATION WINDOW SIZE ***
CALL CWDSZE
C
C
C
*** OPEN THE CORRELATION WINDOW SCRATCH FILE ***
CALL OPNCWF(INTERR)
IF(INTERR.EQ.1)RETURN
C
C
C
*** SELECT THE INTEREST OPERATOR FOR PROCESSING ***
10 CALL IOPSEL(INT,INTERR,INTEND)
IF(INTERR.EQ.1)RETURN
IF(INTEND.EQ.1)RETURN
IF(ANSWER.EQ.'NO')GO TO 20
C
C
C
C
*** ENTER THE LOGICAL UNIT NUMBER AND FILE NAME FOR STORING
THE INTERESTING POINTS ***
TYPE *, ' '
TYPE *, 'PLEASE ENTER THE LOGICAL UNIT NUMBER AND FILE NAME FOR THE'
TYPE *, 'FILE YOU WISH TO USE TO STORE THE INTERESTING POINTS:'
CALL IF1FLU
C
C
C
*** OUTPUT THE INTERESTING POINTS ***
CALL IPSOUT(INTERR)
IF(INTERR.EQ.1)RETURN
C
C
C
*** TEST FOR LISTING ***
20 TYPE *, ' '
TYPE *, 'DO YOU WISH TO LIST THE INTERESTING POINTS ON THE'
TYPE *, 'LINE PRINTER?'
TYPE *, 'IF YOU DO, TYPE YES. OTHERWISE, TYPE NO.'
TYPE *, ' '
TYPE 1
ACCEPT 2,ANSWER
CALL ANSCHK(ANSWER)
IF(ANSWER.EQ.'NO')RETURN
C
C
C
C
*** LIST THE INTERESTING POINTS ***
CALL IPLST
C
RETURN
END

```

```

SUBROUTINE WDSIZE
C
C THIS SUBROUTINE ALLOWS THE USER TO DETERMINE THE NUMBER OF
C ROWS AND COLUMNS OF THE WINDOW TO BE MOVED ACROSS THE IMAGE.
C FROM THIS, THE WINDOW SIZE AND ITS INVERSE MAY BE OBTAINED.
C THE USER IS ASKED TO CHECK THE INPUT INFORMATION AND MAKE
C CORRECTIONS IF NECESSARY.
C
COMMON /WINDOW/ WNDROW,WNDCOL,IX,IY,WDSIZE,WNDINV
INTEGER WNDROW,WNDCOL,WDSIZE
TYPE *,' '
TYPE *,'PLEASE TYPE THE NUMBER OF ROWS AND COLUMNS'
TYPE *,'OF THE WINDOW:'
TYPE *,' '
C
C *** ENTER THE NUMBER OF ROWS AND COLUMNS OF THE WINDOW ***
C
CALL WRCFND
C
C ***COMPUTE THE WINDOW SIZE (NUMBER OF ROWS X NUMBER OF COLUMNS)***
C
WDSIZE = WNDROW*WNDCOL
C
C *** COMPUTE THE INVERSE OF THE WINDOW SIZE ***
C
WNDINV = 1.0/WDSIZE
C
WRITE (6,*) ' '
WRITE (6,*) '*** WINDOW INFORMATION ***'
WRITE (6,*) ' '
WRITE (6,*) 'NUMBER OF WINDOW ROWS = ',WNDROW
WRITE (6,*) 'NUMBER OF WINDOW COLUMNS = ',WNDCOL
WRITE (6,*) 'WINDOW SIZE (ROWSXCOLUMNS) = ',WDSIZE
WRITE (6,*) 'INVERSE OF WINDOW SIZE = ',WNDINV
WRITE (6,*) ' '
RETURN
END

```

```

SUBROUTINE WRCFND
C
C THIS SUBROUTINE ALLOWS THE USER TO ENTER THE NUMBER OF ROWS AND
C COLUMNS INTERACTIVELY FOR THE WINDOW TO BE MOVED ACROSS THE IM-
C AGE. THE USER MAY CORRECT THIS INFORMATION IF AN ENTRY ERROR
C OCCURS.
C
C *** ENTER THE NUMBER OF ROWS AND COLUMNS ***
C
CALL WDROW
CALL WDCOL
C
C *** OPTIONAL ENTRY INFORMATION CHECK ***
C
TYPE *, ' '
TYPE *, 'DO YOU WISH TO CHECK FOR ENTRY ERRORS?'
TYPE *, 'IF YOU DO, TYPE YES. OTHERWISE, TYPE NO.'
TYPE *, ' '
TYPE 1
1 FORMAT(5X, 'ANSWER = ', S)
ACCEPT 2, ANSWER
2 FORMAT(A3)
C
C *** CHECK ANSWER ENTRY ***
C
CALL ANSCHK(ANSWER)
C
IF(ANSWER.EQ.'NO')GO TO 3
C
C *** CHECK ENTRY INFORMATION ***
C
CALL WRCCHK
C
3 RETURN
END

```

```

SUBROUTINE WRCCHK
C
C THIS SUBROUTINE ALLOWS THE USER TO CHECK THE INFORMATION
C INPUT FOR THE NUMBER OF ROWS AND COLUMNS OF THE WINDOW, AND
C TO CORRECT THEM IF DESIRED.
C
COMMON /WINDOW/ WNDROW,WNDCOL
INTEGER WNDROW,WNDCOL
1 TYPE *,' '
TYPE *,'HERE IS WHAT YOU HAVE INPUT:'
TYPE *,' '
C
C *** DISPLAY THE NUMBER OF WINDOW ROWS AND COLUMNS ***
C
TYPE *,'NUMBER OF WINDOW ROWS = ',WNDROW
TYPE *,'NUMBER OF WINDOW COLUMNS = ',WNDCOL
C
C *** VERIFY THE CORRECT RESPONSE ***
C
CALL VERIFY(ANSWER)
C
IF(ANSWER.EQ.'YES')GO TO 2
C
C *** CHANGE INCORRECT INFORMATION ***
C
CALL WRCCOR
C
GO TO 1
2 RETURN
END

```

```

SUBROUTINE WRCCOR
C
C THIS SUBROUTINE ALLOWS THE USER TO CORRECT THE INFORMATION
C INPUT FOR THE NUMBER OF WINDOW ROWS AND COLUMNS.
C
REAL ITEM
C
C *** MAKE ERROR CORRECTIONS ***
C
TYPE *, ' '
TYPE *, 'DO YOU WANT TO CHANGE THE NUMBER OF ROWS, COLUMNS, '
TYPE *, 'OR BOTH?'
1 TYPE *, 'IF YOU WANT TO CHANGE THE NUMBER OF ROWS, TYPE ROWS.'
TYPE *, 'IF YOU WANT TO CHANGE THE NUMBER OF COLUMNS, TYPE COLS.'
TYPE *, 'IF YOU WANT TO CHANGE BOTH, TYPE BOTH.'
TYPE *, ' '
TYPE 2
2 FORMAT(5X, 'ITEM YOU WISH TO CHANGE = ', $)
ACCEPT 3, ITEM
3 FORMAT(A4)
IF(ITEM.EQ.'BOTH')GO TO 10
IF(ITEM.EQ.'ROWS')GO TO 20
IF(ITEM.EQ.'COLS')GO TO 30
TYPE *, ' '
TYPE *, '*** ITEM ENTRY ERROR - PLEASE RETYPE ***'
TYPE *, ' '
GO TO 1
C
C *** CHANGE BOTH NUMBER OF ROWS AND COLUMNS ***
C
10 CALL WDROW
CALL WDCOL
RETURN
C
C *** CHANGE NUMBER OF ROWS ***
C
20 CALL WDROW
RETURN
C
C *** CHANGE NUMBER OF COLUMNS ***
C
30 CALL WDCOL
RETURN
END

```

```

SUBROUTINE WROW
C
C THIS SUBROUTINE ALLOWS THE USER TO INPUT THE NUMBER OF ROWS
C OF THE WINDOW TO BE MOVED ACROSS THE IMAGE.
C
INCLUDE 'COMMON.FTN'
COMMON /WINDOW/ WNDROW
INTEGER WNDROW
I = 2*N - 1
C
C *** ENTER THE NUMBER OF WINDOW ROWS ***
C
TYPE 1
FORMAT(5X,'NUMBER OF WINDOW ROWS = ',S)
ACCEPT *,WNDROW
IF(WNDROW.GT.1)GO TO 2
IF(WNDROW.LT.2)GO TO 3
C
RETURN
2 TYPE *,' '
TYPE *,'*** MAXIMUM WINDOW ROW SIZE',I,' EXCEEDED IN WROW ***'
GO TO 4
3 TYPE *,' '
TYPE *,'*** ROW SIZE SMALLER THAN 2 (MINIMUM) ATTEMPTED ***'
C
C *** CORRECT ROW SIZE ***
C
CALL WRCCOR
C
IF(WNDROW.GT.1)GO TO 2
IF(WNDROW.LT.2)GO TO 3
RETURN
END

```



```

C      SUBROUTINE WDCOL
C
C      THIS SUBROUTINE ALLOWS THE USER TO INPUT THE NUMBER OF COLUMNS
C      OF THE WINDOW TO BE MOVED ACROSS THE IMAGE.
C
C      COMMON /IMAGE/ X,IMGCOL
C      COMMON /WINDOW/ XX,WNDCOL
C      INTEGER WNDCOL,X,XX
C      N = IMGCOL - 1
C
C      *** ENTER THE NUMBER OF WINDOW COLUMNS ***
C
C      TYPE 1
1      FORMAT(5X,'NUMBER OF WINDOW COLUMNS = ',S)
C      ACCEPT *,WNDCOL
C      IF(WNDCOL.GT.N)GO TO 2
C      IF(WNDCOL.LT.2)GO TO 3
C
C      RETURN
2      TYPE *,' '
C      TYPE *,'*** MAXIMUM WINDOW COLUMN SIZE',N,' EXCEEDED IN WDCOL ***'
C      GO TO 4
3      TYPE *,' '
C      TYPE *,'*** COLUMN SIZE LESS THAN 2 (MINIMUM) ATTEMPTED ***'
C
C      *** CORRECT COLUMN SIZE ***
C
C      CALL WRCCHK
C
C      IF(WNDCOL.GT.N)GO TO 2
C      IF(WNDCOL.LT.2)GO TO 3
C      RETURN
C      END

```

```

C      SUBROUTINE IMGIN(INT,IMGERR)
C
C      THIS SUBROUTINE INPUTS IMAGE INTENSITY DATA INTO THE INTENSITY
C      ARRAY ONE ROW AT A TIME.  THE NUMBER OF ROWS INPUT IS EQUAL TO
C      THE NUMBER OF WINDOW ROWS SPECIFIED.  THE I/O BUFFER IS THEN
C      FILLED WITH L MORE ROWS OF INTENSITY DATA.  THE LOGICAL UNIT
C      NUMBER AND FILENAME OF THE IMAGE TO BE INPUT ARE ENTERED INTER-
C      ACTIVELY, AND THE USER MAY CORRECT THEM IF AN ENTRY ERROR OCCURS.
C
C      INCLUDE 'COMMON.FTN'
C      VIRTUAL INT(2*N,M)
C      COMMON /BUF/ IOBUFR(L,M)
C      COMMON /IMAGE/ IMGROW,IMGCOL
C      COMMON /WINDOW/ WNDROW
C      COMMON /COUNT/ X,XX,RECNUM
C      LOGICAL*1 IOBUFR
C      INTEGER RECNUM,WNDROW,X,XX
C      WRITE (6,*) '*** ENTER IMGIN ***'
C
C      *** ENTER THE LOGICAL UNIT NUMBER AND FILE NAME ***
C
C      TYPE *, ' '
C      TYPE *, 'PLEASE ENTER THE LOGICAL UNIT NUMBER AND FILE NAME FOR'
C      TYPE *, 'THE FILE CONTAINING THE IMAGE TO BE PROCESSED:'
C      CALL FLUFND
C
C      *** OPEN THE FILE DEFINED BY THE LOGICAL UNIT NUMBER AND FILENAME
C      PREVIOUSLY ENTERED ***
C
C      CALL OPNOLD(IMGERR)
C      IF(IMGERR.EQ.1)RETURN
C
C      DO 30 RECNUM=1,WNDROW
C
C      *** READ THE IMAGE INTENSITY DATA INTO THE IO BUFFER ***
C
C      I = 1
C      CALL INPUT(RECNUM,I,IMGERR)
C      IF(IMGERR.EQ.1)RETURN
C
C      *** PERFORM BYTE TO WORD NUMBER CONVERSION ***
C
C      DO 10 J=1,IMGCOL
C      INT(RECNUM,J) = IOBUFR(1,J)
C      IF(INT(RECNUM,J).LT.0)INT(RECNUM,J) = INT(RECNUM,J) + 256
C      CONTINUE
10
C
C      *** WRITE TO LINE PRINTER ***
C

```

```

D      WRITE (6,*) 'RECORD NUMBER',RECNUM,' : '
D      WRITE (6,15)
D15    FORMAT(1X,24('-'))
D      WRITE (6,20)(INT(RECNUM,J),J=1,IMGCOL)
D20    FORMAT(26(2X,I3))
D      WRITE (6,*) ' '
30     CONTINUE
C
C      *** FILL I/O BUFFER ***
C
C      CALL BUFRIN(IMGERR)
C
C      RETURN
C      END

```

```

SUBROUTINE CWDSE
C
C THIS SUBROUTINE ALLOWS THE USER TO DETERMINE THE NUMBER OF
C ROWS AND COLUMNS OF THE CORRELATION WINDOW ASSOCIATED WITH
C EACH INTERESTING POINT. THE USER IS THEN ASKED TO CHECK
C THE INPUT INFORMATION AND MAKE CORRECTIONS IF NECESSARY.
C
COMMON /CORWND/ CWDROW,CWDCOL,CWDRWH,CWDCLH,CWSZE,CWDINV
INTEGER CWDROW,CWDCOL,CWDRWH,CWDCLH,CWSZE
TYPE *,' '
TYPE *,'PLEASE TYPE THE NUMBER OF ROWS AND COLUMNS'
TYPE *,'OF THE CORRELATION WINDOW:'
TYPE *,' '
C
C *** ENTER THE NUMBER OF ROWS AND COLUMNS OF THE WINDOW ***
C
CALL CRCFND
C
C ***COMPUTE THE WINDOW SIZE (NUMBER OF ROWS X NUMBER OF COLUMNS)***
C
CWSZE = CWDROW*CWDCOL
CWDRWH = CWDROW/2 + 1
CWDCLH = CWDCOL/2 + 1
C
C *** COMPUTE THE INVERSE OF THE WINDOW SIZE ***
C
CWDINV = 1.0/CWSZE
C
WRITE (6,*) ' '
WRITE (6,*) '*** CORRELATION WINDOW INFORMATION ***'
WRITE (6,*) ' '
WRITE (6,*) 'NUMBER OF CORRELATION WINDOW ROWS = ',CWDROW
WRITE (6,*) 'NUMBER OF CORRELATION WINDOW COLUMNS = ',CWDCOL
WRITE (6,*) 'CORRELATION WINDOW SIZE (ROWSXCOLUMNS) = ',CWSZE
WRITE (6,*) 'INVERSE OF CORRELATION WINDOW SIZE = ',CWDINV
WRITE (6,*) ' '
RETURN
END

```

```

C      SUBROUTINE CRCFND
C
C      THIS SUBROUTINE ALLOWS THE USER TO ENTER THE NUMBER OF ROWS AND
C      COLUMNS INTERACTIVELY FOR THE CORRELATION WINDOW ASSOCIATED WITH
C      EACH INTERESTING POINT.  THE USER MAY CORRECT THIS INFORMATION
C      IF AN ENTRY ERROR OCCURS.
C
C      *** ENTER THE NUMBER OF ROWS AND COLUMNS ***
C
C      CALL CWROW
C      CALL CWCOL
C
C      *** OPTIONAL ENTRY INFORMATION CHECK ***
C
C      TYPE *, ' '
C      TYPE *, 'DO YOU WISH TO CHECK FOR ENTRY ERRORS?'
C      TYPE *, 'IF YOU DO, TYPE YES.  OTHERWISE, TYPE NO.'
C      TYPE *, ' '
C      TYPE 1
1      FORMAT(5X, 'ANSWER = ', $)
2      ACCEPT 2, ANSWER
C      FORMAT(A3)
C
C      *** CHECK ANSWER ENTRY ***
C
C      CALL ANSCHK(ANSWER)
C
C      IF(ANSWER.EQ.'NO')GO TO 3
C
C      *** CHECK ENTRY INFORMATION ***
C
C      CALL CRCCHK
C
3      RETURN
      END

```

```

SUBROUTINE CRCCHK
C
C THIS SUBROUTINE ALLOWS THE USER TO CHECK THE INFORMATION
C INPUT FOR THE NUMBER OF ROWS AND COLUMNS OF THE CORRELATION
C WINDOW, AND TO CORRECT THEM IF DESIRED.
C
COMMON /CORWND/ CWDROW,CWDCOL
INTEGER CWDROW,CWDCOL
1 TYPE *, ' '
TYPE *, 'HERE IS WHAT YOU HAVE INPUT:'
TYPE *, ' '
C
C *** DISPLAY THE NUMBER OF WINDOW ROWS AND COLUMNS ***
C
TYPE *, 'NUMBER OF CORRELATION WINDOW ROWS = ', CWDROW
TYPE *, 'NUMBER OF CORRELATION WINDOW COLUMNS = ', CWDCOL
C
C *** VERIFY THE CORRECT RESPONSE ***
C
CALL VERIFY(ANSWER)
C
IF(ANSWER.EQ.'YES')GO TO 2
C
C *** CHANGE INCORRECT INFORMATION ***
C
CALL CRCCOR
C
GO TO 1
2 RETURN
END

```

```

SUBROUTINE CRCCOR
C
C THIS SUBROUTINE ALLOWS THE USER TO CORRECT THE INFORMATION
C INPUT FOR THE NUMBER OF CORRELATION WINDOW ROWS AND COLUMNS.
C
REAL ITEM
C
C *** MAKE ERROR CORRECTIONS ***
C
TYPE *, ' '
TYPE *, 'DO YOU WANT TO CHANGE THE NUMBER OF ROWS, COLUMNS,'
TYPE *, 'OR BOTH?'
1 TYPE *, 'IF YOU WANT TO CHANGE THE NUMBER OF ROWS, TYPE ROWS.'
TYPE *, 'IF YOU WANT TO CHANGE THE NUMBER OF COLUMNS, TYPE COLS.'
TYPE *, 'IF YOU WANT TO CHANGE BOTH, TYPE BOTH.'
TYPE *, ' '
TYPE 2
2 FORMAT(5X, 'ITEM YOU WISH TO CHANGE = ', $)
ACCEPT 3, ITEM
3 FORMAT(A4)
IF (ITEM.EQ. 'BOTH') GO TO 10
IF (ITEM.EQ. 'ROWS') GO TO 20
IF (ITEM.EQ. 'COLS') GO TO 30
TYPE *, ' '
TYPE *, '*** ITEM ENTRY ERROR - PLEASE RETYPE ***'
TYPE *, ' '
GO TO 1
C
C *** CHANGE BOTH NUMBER OF ROWS AND COLUMNS ***
C
10 CALL CWROW
CALL CWCOL
RETURN
C
C *** CHANGE NUMBER OF ROWS ***
C
20 CALL CWROW
RETURN
C
C *** CHANGE NUMBER OF COLUMNS ***
C
30 CALL CWCOL
RETURN
END

```

```

C      SUBROUTINE CWROW
C
C      THIS SUBROUTINE ALLOWS THE USER TO INPUT THE NUMBER OF ROWS
C      OF THE CORRELATION WINDOW.
C
C      INCLUDE 'COMMON.FTN'
C      COMMON /CORWND/ CWDROW
C      INTEGER CWDROW
C
C      *** ENTER THE NUMBER OF WINDOW ROWS ***
C
C      TYPE 1
1      FORMAT(5X,'NUMBER OF CORRELATION WINDOW ROWS = ',S)
      ACCEPT *,CWDROW
      IF(CWDROW.GT.M/6)GO TO 2
      IF(CWDROW.LT.2)GO TO 3
C
C      RETURN
2      TYPE *,' '
      TYPE *,'*** MAXIMUM WINDOW ROW SIZE',M/6,' EXCEEDED IN CWROW ***'
      GO TO 4
3      TYPE *,' '
      TYPE *,'*** ROW SIZE SMALLER THAN 2 (MINIMUM) ATTEMPTED ***'
C
C      *** CORRECT ROW SIZE ***
C
C      CALL CRCCOR
C
C      IF(CWDROW.GT.M/6)GO TO 2
C      IF(CWDROW.LT.2)GO TO 3
C      RETURN
C      END
C      SUBROUTINE CWCOL
C
C      THIS SUBROUTINE ALLOWS THE USER TO INPUT THE NUMBER OF COLUMNS
C      OF THE CORRELATION WINDOW.
C
C      INCLUDE 'COMMON.FTN'
C      COMMON /CORWND/ CWDROW,CWDCOL
C      INTEGER CWDROW,CWDCOL
C
C      *** ENTER THE NUMBER OF WINDOW COLUMNS ***
C
C      TYPE 1
1      FORMAT(5X,'NUMBER OF CORRELATION WINDOW COLUMNS = ',S)
      ACCEPT *,CWDCOL
      I = CWDROW*CWDCOL
      J = M/(3*CWDROW)
      IF(I.GT.M/3)GO TO 2

```



```

IF(CWDCOL.LT.2)GO TO 3
C
RETURN
2
TYPE *, ' '
TYPE *, '*** MAXIMUM WINDOW COLUMN SIZE',J,' EXCEEDED IN CWCOL ***'
TYPE *, 'PRESENT WINDOW COLUMN SIZE = ',CWDCOL
TYPE *, ' '
TYPE *, 'PRODUCT OF ROWS AND COLUMNS CANNOT EXCEED:',M/3
TYPE *, 'PRESENT PRODUCT OF ROWS AND COLUMNS IS:',I
GO TO 4
3
TYPE *, ' '
TYPE *, '*** COLUMN SIZE LESS THAN 2 (MINIMUM) ATTEMPTED ***'
C
C
C
4
C
CALL CRCCHK
I = CWDROW*CWDCOL
J = M/(3*CWDROW)
IF(I.GT.M/3)GO TO 2
IF(CWDCOL.LT.2)GO TO 3
RETURN
END

```

```

SUBROUTINE OPNCWF(OPNERR)
C
C THIS SUBROUTINE OPENS A SCRATCH FILE FOR TEMPORARILY STORING
C THE CORRELATION WINDOW SURROUNDING EACH INTERESTING POINT. THE
C FILE IS A SEQUENTIAL FORMATTED FILE WITH THE DEFAULT FILENAME.
C
COMMON /FILE/ LUN
COMMON /CWFILE/ LUNCW
COMMON /CORWND/ IA,IB,IC,ID,CWDSZE
INTEGER OPNERR,CWDSZE
C
C *** DEFINE THE LOGICAL UNIT NUMBER ***
C
LUNCW = LUN + 1
IF(LUNCW.EQ.5)LUNCW = 7
IF(LUNCW.GT.10)LUNCW = 1
C
C *** OPEN THE FILE ***
C
OPEN(UNIT=LUNCW,TYPE='SCRATCH',RECORDSIZE=3*CWDSZE+4,ERR=100)
C
RETURN
100 TYPE *, '
TYPE *, '*** OPEN ERROR OCCURED IN OPNCWF ON LOGICAL UNIT',LUNCW,
1' ***'
CLOSE(UNIT=LUNCW,DISPOSE='DELETE')
CALL CLOSKP(OPNERR)
OPNERR = 1
WRITE (6,*) '*** EXIT OPNCWF - OPEN ERROR TERMINATION ***'
RETURN
END

```

```

SUBROUTINE IOPSEL(INT,IOPERR,IOPEND)

C
C THIS SUBROUTINE ALLOWS THE USER TO SELECT THE INTEREST
C OPERATOR TO BE USED TO FIND THE INTERESTING POINTS OF
C THE SELECTED IMAGE.
C

COMMON /IOP/ IOPNBR
1  TYPE *, '
TYPE *, 'ENTER THE NUMBER CORRESPONDING TO THE INTEREST OPERATOR'
TYPE *, 'YOU WISH TO USE TO FIND THE INTERESTING POINTS OF THE'
TYPE *, 'SELECTED IMAGE:'
TYPE *, '
TYPE *, '    1. SIMPLE VARIANCE'
TYPE *, '    2. DIRECTED VARIANCE'
TYPE *, '    3. EDGED VARIANCE'
TYPE *, '    4. NONE YET'
TYPE *, '    5. NONE YET'
TYPE *, '    6. NONE YET'
TYPE *, '    7. NONE YET'
TYPE *, '    8. NONE YET'
TYPE *, '    9. NONE YET'
TYPE *, '   10. NONE YET'
TYPE *, '   11. TERMINATE INTERESTING POINT PROCESSING'
TYPE *, '   12. TERMINATE PROGRAM EXECUTION'
TYPE *, '
TYPE 5
5  FORMAT(5X,' INTEREST OPERATOR NUMBER = ',S)
ACCEPT *, IOPNBR
IF(IOPNBR.LT.1.OR.IOPNBR.GT.12)GO TO 200
GO TO(10,20,30,40,50,60,70,80,90,100,110,120)IOPNBR
10 CALL IOP1(INT,IOPERR)
RETURN
20 CALL IOP2(INT,IOPERR)
RETURN
30 CALL IOP3(INT,IOPERR)
RETURN
40 CALL IOP4(IOPERR)
GO TO 1
50 CALL IOP5(IOPERR)
GO TO 1
60 CALL IOP6(IOPERR)
GO TO 1
70 CALL IOP7(IOPERR)
GO TO 1
80 CALL IOP8(IOPERR)
GO TO 1
90 CALL IOP9(IOPERR)
GO TO 1

```

100 CALL IOPI0(IOPERR)
GO TO 1
110 CALL CLOSKP(IOPERR)
RETURN
120 IOPEND = 1
CALL CLOSKP(IOPEPR)
RETURN
200 TYPE *, ' '
TYPE *, '*** INVALID INTEREST OPERATOR NUMBER', IOPNBR, ' ATTEMPTED
1***'
GO TO 1
END

```

SUBROUTINE IF1FLU
C
C THIS SUBROUTINE ALLOWS THE USER TO ENTER THE LOGICAL UNIT
C NUMBER AND FILE NAME INTERACTIVELY FOR INTERESTING POINT
C SET INPUT/OUTPUT OPERATIONS. THE USER MAY CORRECT THIS
C INFORMATION IF AN ENTRY ERROR OCCURS.
C
COMMON /IFILE1/ LUN1,NCHAR1,FNAME1(34)
LOGICAL*1 FNAME1
C
C *** ENTER THE LOGICAL UNIT NUMBER AND FILE NAME ***
C
CALL LUNBR1
CALL FILNM1
C
C *** OPTIONAL ENTRY INFORMATION CHECK ***
C
TYPE *, ' '
TYPE *, 'DO YOU WISH TO CHECK FOR ENTRY ERRORS?'
TYPE *, 'IF YOU DO, TYPE YES. OTHERWISE, TYPE NO.'
TYPE *, ' '
TYPE 1
1 FORMAT(5X, 'ANSWER = ', S)
ACCEPT 2, ANSWER
2 FORMAT(A3)
C
C *** CHECK ANSWER ENTRY ***
C
CALL ANSCHK(ANSWER)
C
IF(ANSWER.EQ.'NO')GO TO 3
C
C *** CHECK ENTRY INFORMATION ***
C
CALL IF1CHK
C
3 WRITE (6,*) ' '
WRITE (6,*) '*** INTERESTING POINT SET 1 FILE INFORMATION ***'
WRITE (6,*) ' '
WRITE (6,*) 'LOGICAL UNIT NUMBER = ', LUN1
WRITE (6,4) (FNAME1(I), I=1, NCHAR1)
4 FORMAT(1X, 'FILENAME = ', <NCHAR1+1>A1)
WRITE (6,*) ' '
RETURN
END

```

```

SUBROUTINE IFCHK
C
C THIS SUBROUTINE ALLOWS THE USER TO CHECK THE INFORMATION
C INPUT FOR THE FILE NAME AND LOGICAL UNIT NUMBER, AND TO
C CORRECT THEM IF DESIRED.
C
COMMON /IFILE1/ LUN1,NCHAR1,FNAME1(34)
LOGICAL*1 FNAME1
1  TYPE *,' '
   TYPE *,'HERE IS WHAT YOU HAVE INPUT:'
   TYPE *,' '
C
C *** DISPLAY THE FILE NAME AND LOGICAL UNIT NUMBER INPUT ***
C
   TYPE *,'LOGICAL UNIT NUMBER = ',LUN1
   TYPE 2,(FNAME1(I),I=1,NCHAR1)
2  FORMAT(1X,'FILENAME = ',<NCHAR1+1>A1)
C
C *** VERIFY THE CORRECT RESPONSE ***
C
   CALL VERIFY(ANSWER)
C
   IF(ANSWER.EQ.'YES')GO TO 3
C
C *** CHANGE INCORRECT INFORMATION ***
C
   CALL IFICOR
C
   GO TO 1
3  RETURN
   END

```

```

SUBROUTINE IFICOR
C
C THIS SUBROUTINE ALLOWS THE USER TO CORRECT THE INFORMATION
C INPUT FOR THE FILE NAME AND LOGICAL UNIT NUMBER.
C
REAL ITEM
C
C *** MAKE ERROR CORRECTIONS ***
C
TYPE *, ' '
TYPE *, 'DO YOU WANT TO CHANGE THE LOGICAL UNIT NUMBER, '
TYPE *, 'FILE NAME, OR BOTH?'
1 TYPE *, 'IF YOU WANT TO CHANGE THE LOGICAL UNIT NUMBER, TYPE LUN.'
TYPE *, 'IF YOU WANT TO CHANGE THE FILE NAME, TYPE FILE.'
TYPE *, 'IF YOU WANT TO CHANGE BOTH, TYPE BOTH.'
TYPE *, ' '
TYPE 2
2 FORMAT(5X, 'ITEM YOU WISH TO CHANGE = ', S)
ACCEPT 3, ITEM
3 FORMAT(A4)
IF(ITEM.EQ.'BOTH')GO TO 10
IF(ITEM.EQ.'LUN')GO TO 20
IF(ITEM.EQ.'FILE')GO TO 30
TYPE *, ' '
TYPE *, '*** ITEM ENTRY ERROR - PLEASE RETYPE ***'
TYPE *, ' '
GO TO 1
C
C *** CHANGE BOTH FILE NAME AND LOGICAL UNIT NUMBER ***
C
10 CALL FILNM1
CALL LUNBRI
RETURN
C
C *** CHANGE LOGICAL UNIT NUMBER ***
C
20 CALL LUNBRI
RETURN
C
C *** CHANGE FILE NAME ***
C
30 CALL FILNM1
RETURN
END

```

```

SUBROUTINE FILNMI
C
C THIS SUBROUTINE ALLOWS THE USER TO INPUT THE FILE NAME OF
C THE FILE BEING USED FOR INTERESTING POINT SET INPUT/OUTPUT
C OPERATIONS.
C
COMMON /IFILE1/ IX,NCHAR1,FNAME1(34)
LOGICAL*1 FNAME1
TYPE *,' '
TYPE *,'WHAT DEVICE/UIC/FILENAME DO YOU WISH TO USE?'
TYPE *,'ENTER ACCORDING TO THE FOLLOWING FORMAT:'
TYPE *,' '
TYPE *,'          DEV:[ UIC ]FILENAME.TYPE:VERSION'
TYPE *,'EXAMPLE:  DK1:[10,20]FILENAME.IMG;3'
TYPE *,' '
C
C *** INPUT THE DESIRED FILE NAME ***
C
TYPE 1
FORMAT(5X,'FILENAME = ',S)
ACCEPT 2,NCHAR1,FNAME1
2
FORMAT(Q,34A1)
FNAME1(NCHAR1+1) = 0
C
RETURN
END

```



```

SUBROUTINE LUNBRI
C
C THIS SUBROUTINE ALLOWS THE USER TO INPUT THE LOGICAL UNIT
C NUMBER ASSOCIATED WITH THE FILE TO BE USED FOR INTERESTING
C POINT SET INPUT/OUTPUT OPERATIONS.
C
COMMON /CWFILE/ LUNCW
COMMON /IFILE1/ LUN1
1  TYPE *, ' '
   TYPE *, 'WHAT LOGICAL UNIT NUMBER YOU WISH TO USE (1-10)?'
   TYPE *, ' '
C
C *** INPUT THE DESIRED LOGICAL UNIT NUMBER ***
C
   TYPE 2
2  FORMAT(5X, 'LOGICAL UNIT NUMBER = ', S)
   ACCEPT *, LUN1
C
C *** CHECK FOR INCORRECT NUMBER ***
C
   IF(LUN1.LT.1.OR.LUN1.GT.10)GO TO 3
   IF(LUN1.EQ.5)GO TO 4
   IF(LUN1.EQ.6)GO TO 5
   IF(LUN1.EQ.LUNCW)GO TO 6
C
   RETURN
3  TYPE *, ' '
   TYPE *, '*** INVALID LOGICAL UNIT NUMBER', LUN1, ' ATTEMPTED ***'
   TYPE *, ' '
   TYPE *, 'LOGICAL UNIT NUMBER MUST BE BETWEEN 1 AND 10.'
   GO TO 1
4  TYPE *, ' '
   TYPE *, 'LOGICAL UNIT 5 IS RESERVED FOR THE USERS TERMINAL.'
   TYPE *, 'PLEASE USE ANOTHER NUMBER!'
   GO TO 1
5  TYPE *, ' '
   TYPE *, 'LOGICAL UNIT 6 IS RESERVED FOR THE LINE PRINTER.'
   TYPE *, 'PLEASE USE ANOTHER NUMBER!'
   GO TO 1
6  TYPE *, ' '
   TYPE *, 'LOGICAL UNIT NUMBER', LUNCW, ' ALREADY IN USE.'
   TYPE *, 'PLEASE USE ANOTHER NUMBER!'
   GO TO 1
END

```

```

SUBROUTINE IPSOUT(IPSERR)
C
C THIS SUBROUTINE OUTPUTS A COMPLETE SET OF INTERESTING POINTS
C FOR PERMENANT STORAGE AFTER INTERESTING POINT PROCESSING IS
C COMPLETE. THE FIRST RECORD CONTAINS THE NUMBER OF INTERESTING
C POINTS, THE TYPE, AND THE CORRELATION WINDOW ROW, COLUMN, AND
C OVERALL SIZE. SUBSEQUENT RECORDS CONTAIN THE ROW AND COLUMN
C LOCATIONS OF THE POINT ON THE IMAGE, THE VALUE OF THE POINT,
C AND THE CORRELATION WINDOW INTENSITY VALUES.
C
INCLUDE 'COMMON.FTN'
COMMON /IOP/ IOPNBR
COMMON /IFILE1/ LUN1
COMMON /CWFILE/ LUNCW
COMMON /IPS/ IPKNT,IPR(M),IPC(M),VALUIP(M),IPWND(M/3)
COMMON /CORWND/ CWDROW,CWDCOL,IX,IY,CWDSZE
INTEGER CWDROW,CWDCOL,CWDSZE
C
C *** OPEN THE FILE FOR INTERESTING POINT STORAGE ***
C
CALL OPNIF1(IPSERR)
IF(IPSERR.EQ.1)RETURN
C
C *** WRITE FIRST RECORD ***
C
WRITE (LUN1,1,ERR=100) IPKNT,IOPNBR,CWDROW,CWDCOL,CWDSZE
1  FORMAT(5I3)
REWIND LUNCW
DO 10 I=1,IPKNT
C
C *** INPUT EACH INTERESTING POINT CORRELATION WINDOW ***
C
READ(LUNCW,2,END=3,ERR=200)(IPWND(J),J=1,CWDSZE)
2  FORMAT(<CWDSZE>I3)
C
C *** WRITE SECOND AND SUCCESSIVE RECORDS ***
C
WRITE(LUN1,4,ERR=100)IPR(I),IPC(I),VALUIP(I),(IPWND(J),J=1,CWDSZE)
4  FORMAT(2I3,F10.3,<CWDSZE>I3)
C
10  CONTINUE
CLOSE(UNIT=LUNCW,DISPOSE='DELETE')
CLOSE(UNIT=LUN1,DISPOSE='KEEP',ERR=300)
RETURN
100 TYPE *,' '
TYPE *,'*** WRITE ERROR OCCURED IN IPSOUT ON LOGICAL UNIT',LUN1,
1 ' ***'
CLOSE(UNIT=LUNCW,DISPOSE='DELETE')
CLOSE(UNIT=LUN1,DISPOSE='KEEP',ERR=300)

```

```

IPSERR = 1
WRITE (6,*) '*** EXIT IPSOUT - WRITE ERROR TERMINATION ***'
RETURN
200 TYPE *, ' '
TYPE *, '*** READ ERROR OCCURED IN IPSOUT ON LOGICAL UNIT', LUNCW,
1 ' ***'
CLOSE(UNIT=LUNCW, DISPOSE='DELETE')
CLOSE(UNIT=LUN1, DISPOSE='KEEP', ERR=300)
IPSERR = 1
WRITE (6,*) '*** EXIT IPSOUT - READ ERROR TERMINATION ***'
RETURN
300 TYPE *, ' '
TYPE *, '*** CLOSE ERROR OCCURED IN IPSOUT ON LOGICAL UNIT', LUN1,
1 ' ***'
IPSERR = 1
WRITE (6,*) '*** EXIT IPSOUT - CLOSE ERROR TERMINATION ***'
RETURN
END

```

```

C      SUBROUTINE OPNIF1(OPNERR)
C
C      THIS SUBROUTINE OPENS A FILE FOR PERMENANTLY STORING THE SETS OF
C      INTERESTING POINTS AFTER THEIR CREATION.  THE FILE IS A SEQUENT-
C      IAL FORMATTED FILE.  IT IS ALSO USED FOR OPENING A FILE TO INPUT
C      THE FIRST SET OF INTERESTING POINTS FOR MATCHING.
C
C      INCLUDE 'COMMON.FTN'
C      COMMON /IFILE1/ LUN1,IX,FNAME1(34)
C      LOGICAL*1 FNAME1
C      INTEGER OPNERR
C
C      *** OPEN THE FILE ***
C
C      OPEN(UNIT=LUN1,NAME=FNAME1,DISPOSE='KEEP',RECORDSIZE=M+20,
C      1    TYPE='UNKNOWN',ERR=100)
C
C      RETURN
100   TYPE *, ' '
      TYPE *, '*** OPEN ERROR OCCURED IN OPNIF1 ON LOGICAL UNIT',LUN1,
      1 ' ***'
      CLOSE(UNIT=LUN1,DISPOSE='KEEP',ERR=200)
      OPNERR = 1
      WRITE (6,*) '*** EXIT OPNIF1 - OPEN ERROR TERMINATION ***'
      RETURN
200   TYPE *, ' '
      TYPE *, '*** CLOSE ERROR OCCURED IN OPNIF1 ON LOGICAL UNIT',LUN1,
      1 ' ***'
      OPNERR = 1
      WRITE (6,*) '*** EXIT OPNIF1 - CLOSE ERROR TERMINATION ***'
      RETURN
      END

```

```

SUBROUTINE IOPI(INT,IOPERR)
C
C   THIS SUBROUTINE COMPUTES THE INTERESTING POINTS OF
C   THE SELECTED IMAGE USING THE METHOD OF SIMPLE VARIANCE.
C   IF THE VARIANCE COMPUTED FOR EACH WINDOW POSITION OF
C   THE IMAGE IS ABOVE THE USER-SPECIFIED THRESHOLD, THAT
C   POINT IS DEFINED TO BE AN INTERESTING POINT.
C
D   WRITE (6,*) '*** ENTER IOPI ***'
C
C   *** SET THRESHOLD VALUE ***
C
C   CALL THRHLD(THRESH)
C
C   *** INITIALIZE INTEREST OPERATOR PARAMETERS ***
C
C   CALL INITAL(IMGEND,IPSEND)
C
C   *** COMPUTE THE SIMPLE VARIANCE OF EACH WINDOW POSITION ***
10  CALL SVAR(INT,VARNCE,IOPERR)
    IF(IOPERR.EQ.1)GO TO 20
C
C   *** TEST COMPUTED VARIANCE TO SEE IF INTERESTING POINT EXISTS ***
C
C   IF(VARNCE.GT.THRESH)CALL IPSTOR(VARNCE,IOPERR,IPSEND)
C   IF(IOPERR.EQ.1.OR.IPSEND.EQ.1)GO TO 20
C
C   *** MOVE WINDOW ONE IMAGE ELEMENT ***
C
C   CALL WNDMOV(IMGEND)
C   IF(IMGEND.NE.1)GO TO 10
C
20  CALL CLOSKP(IOPERR)
    RETURN
    END

```

```

SUBROUTINE INITAL(IMGEND,IPSEND)
C
C   THIS SUBROUTINE INITIALIZES THE WINDOW POSITION AND VARIOUS
C   OTHER PARAMETERS USED IN THE INTEREST OPERATOR SELECTED.
C
COMMON /IPS/ IPKNT
COMMON /COUNT/ WNDKNT
COMMON /MOVE/ ROWKNT, COLKNT, ROWDIF, COLDIF
COMMON /WINDOW/ WNDROW, WNDCOL, WNDRWH, WNDCLH
INTEGER ROWKNT, COLKNT, ROWDIF, COLDIF
INTEGER WNDROW, WNDCOL, WNDRWH, WNDCLH, WNDKNT
C
C   *** INITIALIZE WINDOW POSITION ***
C
ROWKNT = 1
COLKNT = 1
WNDKNT = 1
ROWDIF = WNDROW
COLDIF = WNDCOL
WNRWH = WNDROW/2
WNCCLH = WNDCOL/2
C
C
IPKNT = 0
IMGEND = 0
IPSEND = 0
D   WRITE (6,*) ' '
D   WRITE (6,*) 'ROWKNT = ', ROWKNT, '      ROWDIF = ', ROWDIF
D   WRITE (6,*) 'COLKNT = ', COLKNT, '      COLDIF = ', COLDIF
D   WRITE (6,*) ' '
RETURN
END

```

```

SUBROUTINE THRHL(THRESH)
C
C THIS SUBROUTINE ALLOWS THE USER TO INPUT A THRESHOLD
C VALUE USED TO TEST FOR INTERESTING POINTS.
C
1 TYPE *, ' '
  TYPE *, 'WHAT VALUE OF THRESHOLD DO YOU WISH TO USE?'
  TYPE *, ' '
  TYPE 2
2 FORMAT(5X, 'INTERESTING POINT THRESHOLD = ', S)
  ACCEPT *, THRESH
  TYPE *, ' '
  TYPE *, 'HERE IS WHAT YOU HAVE INPUT:'
  TYPE *, ' '
C
C *** DISPLAY THE VALUE OF THRESHOLD ***
C
  TYPE *, 'INTERESTING POINT THRESHOLD = ', THRESH
C
C *** VERIFY THE CORRECT RESPONSE ***
C
  CALL VERIFY(ANSWER)
C
  IF(ANSWER.EQ.'NO')GO TO 1
  WRITE (6,*) ' '
  WRITE (6,*) '*** THRESHOLD FOR INTERESTING POINT TESTS ***'
  WRITE (6,*) ' '
  WRITE (6,*) 'INTERESTING POINT THRESHOLD = ', THRESH
  WRITE (6,*) ' '
  RETURN
END

```

```

SUBROUTINE SVAR(INT,VARNCE,VARERR)
C
C THIS SUBROUTINE CALCULATES THE SIMPLE VARIANCE OF THE
C DISTRIBUTION OF IMAGE ELEMENT INTENSITIES OVER EACH
C WINDOW POSITION ON THE SELECTED IMAGE.
C
COMMON /MEAN/ X,SQRSUM,MEAN
COMMON /WINDOW/ IA,IB,IC,ID,IE,WNDINV
INTEGER VARERR
REAL MEAN
D WRITE (6,*) '*** ENTER SVAR ***'
C
C *** CALCULATE THE MEAN AND SUM OF SQUARES OF THE ELEMENTS OVER
C EACH WINDOW POSITION ON THE IMAGE ***
C
CALL MEANFD(INT,VARERR)
IF(VARERR.EQ.1)RETURN
C
C *** COMPUTE THE VARIANCE OVER THE WINDOW ***
C
SQRMN = MEAN*MEAN
VARNCE = WNDINV*SQRSUM - SQRMN
C
D WRITE (6,*) 'VARIANCE = ',VARNCE
D WRITE (6,*) ' '
RETURN
END

```



```

SUBROUTINE MEANFD(INT,MNERR)

C
C THIS SUBROUTINE CALCULATES THE MEAN OF ALL THE INTENSITY
C ELEMENTS OVER EACH WINDOW POSITION ON THE SELECTED IMAGE.
C THE SUM OF THE SQUARES OF EACH INTENSITY ELEMENT OVER THE
C WINDOW IS ALSO COMPUTED FOR USE IN CALCULATING THE VARIANCE
C IN SUBROUTINE SVAR. THE MEAN IS CALCULATED BY FIRST SUMMING
C ALL THE ELEMENTS IN A COLUMN THE WIDTH OF THE WINDOW. EACH
C COLUMN IS SUMMED FROM LEFT TO RIGHT ACROSS THE LENGTH OF THE
C ENTIRE IMAGE. AFTER THIS, FOR EACH WINDOW POSITION, THE
C COLUMNS FALLING WITHIN THE WINDOW ARE SUMMED TO OBTAIN THE
C SUM OF ALL THE INTENSITY ELEMENTS WITHIN THE WINDOW. THIS
C IS WHEN THE WINDOW IS AT THE LEFTMOST POSITION ON THE IMAGE.
C THIS SUM IS THEN MULTIPLIED BY THE INVERSE OF THE WINDOW
C SIZE TO OBTAIN THE MEAN. WHEN THE WINDOW IS MOVED TO THE
C RIGHT, THE COLUMN TO THE LEFT IS SUBTRACTED AND THE COLUMN
C TO THE RIGHT IS ADDED TO THE PREVIOUS ELEMENT SUM TO FORM
C THE NEW SUM. THIS SUM IS AGAIN MULTIPLIED BY THE INVERSE
C OF THE WINDOW SIZE TO OBTAIN THE MEAN. THE SUM OF THE SQUARES
C OF THE WINDOW ELEMENTS IS OBTAINED IN THE SAME MANNER AS THE
C SUM OF THE ELEMENTS AS DESCRIBED ABOVE. HOWEVER, EACH
C ELEMENT IS MULTIPLIED BY ITSELF FIRST BEFORE IT IS SUMMED.
C

INCLUDE 'COMMON.FTN'
COMMON /REG1/ S(M),T(M)
COMMON /MEAN/ COLSUM,SQRSUM,MEAN
COMMON /MOVE/ ROWKNT,COLKNT,IA,COLDIF
COMMON /WINDOW/ IB,WNDCOL,IC,ID,IE,WNDINV
INTEGER COLKNT,ROWKNT,COLDIF,WNDCOL
REAL MEAN
D WRITE (6,*) '*** ENTER MEANFD ***'
IF(COLKNT.EQ.1)GO TO 10

C
C *** FORM NEW SUM BY SUBTRACTING PREVIOUS COLUMN AND ADDING NEXT
C COLUMN TO THE RIGHT ***
C
COLSUM = COLSUM + S(COLDIF) - S(COLKNT-1)
SQRSUM = SQRSUM + T(COLDIF) - T(COLKNT-1)
D WRITE (6,*) ' '
D WRITE (6,*) 'COLSUM = ',COLSUM,' SQRSUM = ',SQRSUM
GO TO 20
10 IF(ROWKNT.GT.1)CALL ROWCHG(INT,MNERR)
IF(MNERR.EQ.1)RETURN

C
C *** SUM THE ELEMENTS OF EACH COLUMN OF WINDOW LENGTH ACROSS
C THE IMAGE ***
C
CALL ADDCOL(INT)
C

```

```

COLSUM = 0
SQRSUM = 0
C
C   *** FORM INITIAL SUM BY SUMMING THE COLUMNS OF ADDED INTENSITY
C   ELEMENTS WHEN THE WINDOW IS IN ITS LEFTMOST POSITION ON THE
C   SELECTED IMAGE ***
C
DO 15 I=1,WNDCOL
COLSUM = COLSUM + S(I)
SQRSUM = SQRSUM + T(I)
15 CONTINUE
D   WRITE (6,*) ' '
D   WRITE (6,*) 'COLSUM = ',COLSUM,'      SQRSUM = ',SQRSUM
C
C   *** COMPUTE THE MEAN ***
C
20 MEAN = WNDINV*COLSUM
C
D   WRITE (6,*) ' '
D   WRITE (6,*) 'MEAN = ',MEAN
D   WRITE (6,*) ' '
RETURN
END

```

```

SUBROUTINE ADDCOL(INT)
C
C   THIS SUBROUTINE SUMS THE ELEMENTS OF A COLUMN THE WIDTH OF
C   THE WINDOW FOR EACH COLUMN OF THE IMAGE AND STORES THESE SUMS
C   IN ARRAY S. THE SQUARES OF THE ELEMENTS ARE ALSO SUMMED IN
C   LIKE MANNER AND STORED IN ARRAY T.
C
  INCLUDE 'COMMON.FTN'
  VIRTUAL INT(2*N,M)
  COMMON /REG1/ S(M),T(M)
  COMMON /WINDOW/ WNDROW
  COMMON /IMAGE/ X,IMGCOL
  INTEGER WNDROW,X
D  WRITE (6,*) '*** ENTER ADDCOL ***'
  DO 10 J=1,IMGCOL
    S(J) = 0
    T(J) = 0
C
C   *** FORM SUMS OF ELEMENTS AND ELEMENT SQUARES FOR EACH COLUMN ***
C
    DO 10 I=1,WNDROW
      FLTINT = FLOAT(INT(I,J))
      S(J) = S(J) + FLTINT
      SQRINT = FLTINT*FLTINT
      T(J) = T(J) + SQRINT
10  CONTINUE
    RETURN
  END

```

```

SUBROUTINE ROWCHG(INT,ROWERR)

C
C   SINCE THE IMAGE BEING PROCESSED IS GENERALLY TOO LARGE TO
C   BE STORED IN COMPUTER MEMORY IN ITS ENTIRETY, IT IS BUFFERED
C   IN IN SEGMENTS AS IT IS BEING PROCESSED.  AN I/O BUFFER IS
C   USED FOR THIS PURPOSE.  THUS, AS THE WINDOW IS MOVED ACROSS
C   THE IMAGE FROM ROW-TO-ROW, THE ROW PREVIOUSLY PROCESSED IS
C   REPLACED WITH A NEW ROW FROM THE I/O BUFFER IN A ROTATING
C   FASHION.  THIS SUBROUTINE PERFORMS THE ROW CHANGE OPERATION
C   AND CALLS FOR MORE I/O BUFFER DATA WHEN THE CURRENT SUPPLY
C   OF BUFFER DATA HAS ALL BEEN UTILIZED.
C

  INCLUDE 'COMMON.FTN'
  VIRTUAL INT(2*N,M)
  COMMON /BUF/ IOBUFR(L,M)
  COMMON /MOVE/ X,XX,ROWDIF
  COMMON /COUNT/ WNDKNT,BUFKNT
  COMMON /IMAGE/ IMGROW,IMGCOL
  COMMON /WINDOW/ WNDROW
  LOGICAL*1 IOBUFR
  INTEGER ROWERR,WNDKNT,BUFKNT,ROWDIF,WNDROW,X,XX
  WRITE (6,*) '*** ENTER ROWCHG ***'
  WRITE (6,*) ' '
  WRITE (6,*) 'WNDKNT = ',WNDKNT,'      BUFKNT = ',BUFKNT
  WRITE (6,*) ' '
  DO 10 J=1,IMGCOL

  C
  C   *** PERFORM ROW CHANGE ***
  C
    INT(WNDKNT,J) = IOBUFR(BUFKNT,J)

  C
  C   *** PERFORM BYTE TO WORD NUMBER CONVERSION ***
  C
    IF(INT(WNDKNT,J).LT.0)INT(WNDKNT,J) = INT(WNDKNT,J) + 256

  C
10  CONTINUE
    IF(WNDKNT.EQ.WNDROW)WNDKNT = 0
    WNDKNT = WNDKNT + 1
    IF(BUFKNT.EQ.L.AND.ROWDIF.LT.IMGROW)GO TO 20
    BUFKNT = BUFKNT + 1
    RETURN

  C
  C   *** REFILL I/O BUFFER WITH NEW IMAGE DATA ***
  C
20  CALL BUFRIN(ROWERR)

  C
    RETURN
  END

```

```

SUBROUTINE BUFRIN(BUFERR)
C
C THIS SUBROUTINE INPUTS IMAGE INTENSITY DATA INTO THE I/O
C BUFFER L ROWS AT A TIME.
C
INCLUDE 'COMMON.FTN'
COMMON /BUF/ IOBUFR(L,M)
COMMON /INBUF/ INTBUF(M)
COMMON /IMAGE/ IMGROW,IMGCOL
COMMON /COUNT/ X,BUFKNT,RECNUM
LOGICAL*1 IOBUFR
INTEGER BUFERR,EDFLAG,RECNUM,BUFKNT,X
D WRITE (6,*) '*** ENTER BUFRIN ***'
  BUFKNT = 1
  EDFLAG = 0
  NEWREC = RECNUM
  LOOP = RECNUM + (L-1)
  IF(LOOP.GE.IMGROW)GO TO 30
5  I = 0
  DO 20 RECNUM = NEWREC,LOOP
    I = I + 1
C
C *** INPUT IMAGE INTENSITY DATA INTO I/O BUFFER ***
C
CALL INPUT(RECNUM,I,BUFERR)
IF(BUFERR.EQ.1)RETURN
C
C *** PERFORM BYTE TO WORD NUMBER CONVERSION ***
C
D DO 10 J=1,IMGCOL
D INTBUF(J) = IOBUFR(I,J)
D IF(INTBUF(J).LT.0)INTBUF(J) = INTBUF(J) + 256
D10 CONTINUE
C
C *** WRITE TO LINE PRINTER ***
C
D WRITE (6,*) 'RECORD NUMBER',RECNUM,' : '
D WRITE (6,14)
D14 FORMAT(1X,24('-'))
D WRITE (6,15)(INTBUF(J),J=1,IMGCOL)
D15 FORMAT(26(2X,I3))
D WRITE (6,*) ' '
C
20 CONTINUE
  IF(EDFLAG.EQ.1)GO TO 40
  RETURN
30 LOOP = IMGROW
  EDFLAG = 1
  GO TO 5

```

```
40      RECNUM = 1
D      WRITE (6,*) '*** EXIT BUFRIN - LAST IMAGE ROW INPUT ***'
D      WRITE (6,*) ' '
      RETURN
      END
```

```

C      SUBROUTINE WNDMOV(IMGEND)
C
C      THIS SUBROUTINE MOVES A WINDOW OF DIMENSION (WNDROW X WNDCOL)
C      ACROSS THE SELECTED IMAGE ONE ELEMENT AT A TIME. WHEN THE
C      EDGE OF THE IMAGE IS REACHED, THE WINDOW IS MOVED DOWN ONE
C      ROW AND REPOSITIONED AT THE LEFT IMAGE BOUNDARY. WINDOW
C      MOVEMENT IS FROM LEFT TO RIGHT AND FROM TOP TO BOTTOM ACROSS
C      THE IMAGE. THE WINDOW REFERENCE POSITION IS THE UPPER LEFT
C      CORNER OF THE WINDOW, AND IS SPECIFIED RELATIVE TO THE IMAGE
C      BY THE VARIABLES ROWKNT AND COLKNT. THE BOTTOM AND RIGHT
C      BOUNDARIES OF THE WINDOW ARE SPECIFIED BY THE VARIABLES
C      ROWDIF AND COLDIF RESPECTIVELY.
C
C      COMMON /IMAGE/ IMGROW,IMGCOL
C      COMMON /WINDOW/ WNDROW,WNDCOL
C      COMMON /MOVE/ ROWKNT,COLKNT,ROWDIF,COLDIF
C      INTEGER WNDROW,WNDCOL,ROWKNT,COLKNT,ROWDIF,COLDIF
D      WRITE (6,*) '*** ENTER WNDMOV ***'
D      WRITE (6,*) ' '
C
C      *** TEST FOR IMAGE BOUNDARY ***
C
C      IF(COLDIF.EQ.IMGCOL)GO TO 50
C      IF(COLDIF.GT.IMGCOL)GO TO 100
C
C      COLDIF = COLKNT + WNDCOL
C      COLKNT = COLKNT + 1
D      WRITE (6,*) 'ROWKNT = ',ROWKNT,'      ROWDIF = ',ROWDIF
D      WRITE (6,*) 'COLKNT = ',COLKNT,'      COLDIF = ',COLDIF
D      WRITE (6,*) ' '
C      RETURN
C
C      *** TEST FOR END OF IMAGE ***
C
50      IF(ROWDIF.EQ.IMGROW.AND.COLDIF.EQ.IMGCOL)GO TO 200
C      IF(ROWDIF.GT.IMGROW)GO TO 150
C
C      ROWDIF = ROWKNT + WNDROW
C      ROWKNT = ROWKNT + 1
C      COLDIF = WNDCOL
C      COLKNT = 1
D      WRITE (6,*) 'ROWKNT = ',ROWKNT,'      ROWDIF = ',ROWDIF
D      WRITE (6,*) 'COLKNT = ',COLKNT,'      COLDIF = ',COLDIF
D      WRITE (6,*) ' '
C      RETURN
100     TYPE *, ' '
C      TYPE *, '*** IMAGE COLUMN BOUNDARY',IMGCOL,' EXCEEDED BY WINDOW'
C      TYPE *, '      COLUMN BOUNDARY',COLDIF,' IN WNDMOV ***'
C      GO TO 50

```

```

150  TYPE *, ' '
      TYPE *, '*** IMAGE ROW BOUNDARY', IMGROW, ' EXCEEDED BY WINDOW'
      TYPE *, '      ROW BOUNDARY', ROWDIF, ' IN WNDMOV ***'
200  IMGEND = 1
D     WRITE (6,*) '*** EXIT WNDMOV - END OF IMAGE REACHED ***'
      RETURN
      END

```



```

C      SUBROUTINE IOP2(INT,IOPERR)
C
C      THIS SUBROUTINE COMPUTES THE INTERESTING POINTS OF
C      THE SELECTED IMAGE USING THE METHOD OF DIRECTED
C      VARIANCE (THE MORAVEC OPERATOR).  THE DIRECTED VARIANCE
C      VALUES ARE STORED IN A TWO DIMENSIONAL ARRAY D(2,*).
C      THIS ARRAY IS SEARCHED FOR THE LOCAL MAXIMA OF THE
C      DIRECTED VARIANCE VALUES.  EACH OF THESE MAXIMA WHICH
C      ARE GREATER THAN A USER-SPECIFIED THRESHOLD VALUE
C      ARE DEFINED TO BE INTERESTING POINTS.
C
C      INCLUDE 'COMMON.FTN'
C      COMMON /DSTOR/ DCOL,EPSLON,D(2,M)
C      COMMON /MOVE/ ROWKNT,COLKNT
C      COMMON /EDVAR/ EFLAG
C      COMMON /IMAGE/ X,INGCOL
C      COMMON /WINDOW/ XX,WNDCOL
C      INTEGER DCOL,EFLAG,ROWKNT,COLKNT,WNDCOL,X,XX
C      WRITE (6,*) '*** ENTER IOP2 ***'
C
C      *** SET THRESHOLD VALUE ***
C
C      CALL THRHLD(THRESH)
C
C      *** DETERMINE THE VALUE OF EPSILON FOR THE TESTS ***
C
C      CALL EPSFND(EPSLON)
C
C      *** COMPUTE THE DIRECTED VARIANCE FOR EACH WINDOW POSITION ***
C
C      *** INITIALIZE INTEREST OPERATOR PARAMETERS ***
C
C      EFLAG = 0
C      CALL INITAL(INGEND,IPSEND)
C      DCOL = INGCOL - WNDCOL + 1
C
C      10 CALL DVAR(INT,IOPERR)
C      IF(IOPERR.EQ.1)GO TO 50
C      IF(COLKNT.LT.DCOL)GO TO 30
C
C      *** FIND THE LOCAL MAXIMA OF THE DIRECTED VARIANCES ***
C
C      CALL MAXDVR
C
C      IF(ROWKNT.EQ.1)GO TO 30
C
C      *** TEST VARIANCE MAXIMA TO SEE IF INTERESTING POINT EXISTS ***
C
C      DO 20 I=1,DCOL

```

```

IF(D(1,I).LE.THRESH)GO TO 20
COLKNT = I
ROWKNT = ROWKNT - 1
CALL IPSTOR(D(1,I),IOPERR,IPSEND)
IF(IOPERR.EQ.1.OR.IPSEND.EQ.1)GO TO 50
ROWKNT = ROWKNT + 1
COLKNT = DCOL
CONTINUE
20
C
C   *** REPLACE FIRST ROW OF ARRAY D WITH SECOND ROW ***
C
CALL DRWCHG
C
C   *** MOVE WINDOW ONE IMAGE ELEMENT ***
C
30 CALL WNDMOV(IMGEND)
IF(IMGEND.NE.1)GO TO 10
C
C   *** TEST LAST VARIANCE ROW FOR INTERESTING POINTS ***
C
DO 40 I=1,DCOL
IF(D(2,I).LE.THRESH)GO TO 40
COLKNT = I
CALL IPSTOR(D(2,I),IOPERR,IPSEND)
IF(IOPERR.EQ.1.OR.IPSEND.EQ.1)GO TO 50
40 CONTINUE
C
50 CALL CLOSKP(IOPERR)
RETURN
END

```

```

SUBROUTINE DVAR(INT,VARERR)

C
C THIS SUBROUTINE CALCULATES THE DIRECTED VARIANCE OF ALL
C THE INTENSITY ELEMENTS OVER EACH WINDOW POSITION ON THE
C SELECTED IMAGE. FOUR VALUES OF VARIANCE ARE CALCULATED
C FOR EACH WINDOW POSITION. THESE FOUR VALUES ARE CALCULATED
C BY ADDING THE SQUARES OF THE DIFFERENCES OF ADJACENT
C ELEMENTS IN FOUR DIFFERENT DIRECTIONS ACROSS THE WINDOW:
C   (1) ADJACENT ON THE SAME ROW (XSUM)
C   (2) ADJACENT ON THE SAME COLUMN (YSUM)
C   (3) ADJACENT ON THE UPPER-TO-LOWER DIAGONAL (ASUM)
C   (4) ADJACENT ON THE LOWER-TO-UPPER DIAGONAL (BSUM)
C THESE SUMS ARE EACH MULTIPLIED BY THE INVERSE OF THE WINDOW
C SIZE TO OBTAIN THE DIRECTED VARIANCE IN EACH DIRECTION. THE
C OVERALL DIRECTED VARIANCE IS DEFINED TO BE THE MINIMUM OF
C THESE FOUR QUANTITIES. THE MINIMUM OF PERPENDICULAR
C RATIOS OF THESE FOUR VARIANCES IS ALSO COMPUTED WHEN
C THE EDGED VARIANCE IS CALCULATED (EFLAG=1).
C
C   INCLUDE 'COMMON.FTN'
C   COMMON /EDVAR/ EFLAG,RTOMIN
C   COMMON /REG2/ X(M),Y(M),A(M),B(M)
C   COMMON /DRVAR/ XSUM,YSUM,ASUM,BSUM
C   COMMON /MOVE/ ROWKNT,COLKNT,IX,COLDIF
C   COMMON /WINDOW/ IA,WNDCOL,IB,IC,ID,WNDINV
C   INTEGER EFLAG,ROWKNT,COLKNT,COLDIF,WNDCOL,VARERR
D   WRITE (6,*) '*** ENTER DVAR ***'
C   IF(COLKNT.EQ.1)GO TO 10
C
C   *** FORM NEW SUM BY SUBTRACTING PREVIOUS COLUMN AND ADDING NEXT
C   COLUMN TO THE RIGHT ***
C
C   XSUM = XSUM + X(COLDIF) - X(COLKNT-1)
C   YSUM = YSUM + Y(COLDIF) - Y(COLKNT-1)
C   ASUM = ASUM + A(COLDIF) - A(COLKNT-1)
C   BSUM = BSUM + B(COLDIF) - B(COLKNT-1)
C
C   WRITE (6,*) ' '
D   WRITE (6,9) XSUM,YSUM,ASUM,BSUM
D9  FORMAT(1X,'XSUM = ',F11.3,2X,'YSUM = ',F11.3,2X,'ASUM = ',F11.3,2X
D   1,'BSUM = ',F11.3,/)
D   GO TO 20
10  IF(EFLAG.EQ.1)GO TO 12
C   IF(ROWKNT.GT.1)CALL ROWCHG(INT,VARERR)
C   IF(VARERR.EQ.1)RETURN
C
C   *** SUM THE ELEMENTS OF EACH COLUMN OF WINDOW LENGTH ACROSS
C   THE IMAGE ***
C

```

```

12      CALL ADDDIF(INT)
C
      XSUM = 0
      YSUM = 0
      ASUM = 0
      BSUM = 0
C
C      *** FORM INITIAL SUM BY SUMMING THE COLUMNS OF ADDED INTENSITY
C      ELEMENTS WHEN THE WINDOW IS IN ITS LEFTMOST POSITION ON
C      THE IMAGE ***
C
      DO 15 I=1,WNDCOL
      XSUM = XSUM + X(I)
      YSUM = YSUM + Y(I)
      ASUM = ASUM + A(I)
      BSUM = BSUM + B(I)
15      CONTINUE
D      WRITE (6,*) ' '
D      WRITE (6,9) XSUM,YSUM,ASUM,BSUM
C
C      *** COMPUTE THE FOUR DIRECTED VARIANCES ***
C
20      DVAR1 = WNDINV*XSUM
      DVAR2 = WNDINV*YSUM
      DVAR3 = WNDINV*ASUM
      DVAR4 = WNDINV*BSUM
C
D      WRITE (6,21) DVAR1,DVAR2,DVAR3,DVAR4
D21     FORMAT(1X,'DVAR1 = ',F10.3,2X,'DVAR2 = ',F10.3,2X,'DVAR3 = '
D      1,F10.3,2X,'DVAR4 = ',F10.3,/)
      IF(EFLAG.EQ.1)GO TO 30
C
C      *** COMPUTE THE OVERALL DIRECTED VARIANCE ***
C
      DIRVAR = MIN(DVAR1,DVAR2,DVAR3,DVAR4)
C
D      WRITE (6,*) 'DIRVAR = ',DIRVAR
D      WRITE (6,*) ' '
C
C      *** STORE THE OVERALL DIRECTED VARIANCE ***
C
      CALL DVSTOR(DIRVAR)
C
      RETURN
C
C      *** CALCULATE PERPENDICULAR RATIOS OF THE VARIANCES ***
C
30      CALL RATIO(DVAR1,DVAR2,DVAR3,DVAR4,RATIO1,RATIO2,RATIO3,RATIO4)
C
C      *** CALCULATE THE MINIMUM OF THESE RATIOS ***

```

```
C      RTOMIN = MIN(RATIO1,RATIO2,RATIO3,RATIO4)
C
D      WRITE (6,*) 'RTOMIN = ',RTOMIN
D      WRITE (6,*) ' '
      RETURN
      END
```

SUBROUTINE MAXDVR

```

C
C   THIS SUBROUTINE DETERMINES THE LOCAL MAXIMA OF THE
C   DIRECTED VARIANCE VALUES COMPUTED IN SUBROUTINE DVAR.
C   THESE VARIANCE VALUES ARE STORED IN ARRAY D(2,*) TWO
C   ROWS AT A TIME. AFTER THE FIRST ROW HAS BEEN STORED,
C   IT IS COMPARED BY ADJACENT ELEMENTS ACROSS THE ROW.
C   AFTER THE SECOND ROW HAS BEEN STORED, THE SEARCH
C   PROCESS IS EXPANDED TO INCLUDE ELEMENT NEIGHBORS ON
C   BOTH ROWS. AS THE ELEMENTS ARE CHECKED, THOSE THAT
C   ARE SMALLER THAN ALL THEIR NEIGHBORS ARE SET NEGATIVE.
C   ELEMENTS REMAINING POSITIVE AFTER THE SEARCH HAS
C   BEEN COMPLETED ARE THE LOCAL MAXIMA.
C
C   INCLUDE 'COMMON.FTN'
C   COMMON /MOVE/ ROWKNT
C   COMMON /DSTOR/ DCOL, EPSLON, D(2,M)
C   INTEGER DCOL, ROWKNT
D   WRITE (6,*) '*** ENTER MAXDVR ***'
C   K = MOD(ROWKNT,2)
C   IF(ROWKNT.GT.1)GO TO 10
C
C   *** PERFORM INITIAL ROW CHECKS ***
C
C   DO 2 I=1,DCOL-1
C   IF((ABS(D(1,I))-D(1,I+1)).GE.EPSLON)GO TO 1
C   IF(D(1,I).LT.0.0)GO TO 2
C   D(1,I) = -D(1,I)
C   GO TO 2
1   IF(D(1,I+1).LT.0.0)GO TO 2
C   D(1,I+1) = -D(1,I+1)
2   CONTINUE
D   WRITE (6,*) ' '
D   WRITE (6,*) 'FIRST D ROW:'
D   WRITE (6,4)
D4  FORMAT(1X,14('-'))
D   WRITE (6,3)(D(1,I),I=1,DCOL)
D3  FORMAT(10(2X,F11.3))
D   WRITE (6,*) ' '
C   RETURN
C
C   *** TEST THE LEFT END THREE ELEMENTS OF D ARRAY ***
C
C   I = 1
C   IF(K.EQ.0)GO TO 11
C   IF((ABS(D(1,1))-D(2,1)).GE.EPSLON)GO TO 30
C   GO TO 15
11  IF((ABS(D(1,1))-D(2,1)).GT.EPSLON)GO TO 30
15  IF(D(1,1).LT.0.0)GO TO 20

```

```

D(1,1) = -D(1,1)
GO TO 40
20 IF(K.EQ.0)GO TO 21
   IF((ABS(D(1,2))-ABS(D(2,1)))>=EPSLON)GO TO 30
   GO TO 25
21 IF((ABS(D(1,2))-ABS(D(2,1)))>EPSLON)GO TO 30
25 IF(D(1,2).LT.0.0)GO TO 40
   D(1,2) = -D(1,2)
   GO TO 40
30 IF(D(2,1).LT.0.0)GO TO 40
   D(2,1) = -D(2,1)
   IF(D(1,1).GT.0.0)GO TO 40
   GO TO 20
C
C   *** TEST MIDDLE ELEMENTS OF D ARRAY FIVE AT A TIME ***
C
40 KOUNT = 0
   I = I + 1
   IF(I.EQ.DCOL)GO TO 100
   IF(K.EQ.0)GO TO 41
   IF((ABS(D(1,I))-D(2,I))>=EPSLON)GO TO 80
   KOUNT = KOUNT + 1
   GO TO 45
41 IF((ABS(D(1,I))-D(2,I))>EPSLON)GO TO 80
   KOUNT = KOUNT + 1
45 IF(D(1,I).LT.0.0)GO TO 50
   D(1,I) = -D(1,I)
   GO TO 40
50 IF(K.EQ.0)GO TO 51
   IF((ABS(D(2,I-1))-ABS(D(2,I)))>=EPSLON)GO TO 80
   KOUNT = KOUNT + 1
   GO TO 55
51 IF((ABS(D(2,I-1))-ABS(D(2,I)))>EPSLON)GO TO 80
   KOUNT = KOUNT + 1
55 IF(D(2,I-1).LT.0.0)GO TO 60
   D(2,I-1) = -D(2,I-1)
   GO TO 70
60 IF(K.EQ.0)GO TO 61
   IF((ABS(D(1,I-1))-ABS(D(2,I)))>=EPSLON)GO TO 80
   KOUNT = KOUNT + 1
   GO TO 65
61 IF((ABS(D(1,I-1))-ABS(D(2,I)))>EPSLON)GO TO 80
   KOUNT = KOUNT + 1
65 IF(D(1,I-1).LT.0.0)GO TO 70
   D(1,I-1) = -D(1,I-1)
70 IF(K.EQ.0)GO TO 71
   IF((ABS(D(1,I+1))-ABS(D(2,I)))>=EPSLON)GO TO 80
   GO TO 75
71 IF((ABS(D(1,I+1))-ABS(D(2,I)))>EPSLON)GO TO 80
75 IF(D(1,I+1).LT.0.0)GO TO 40

```

```

      D(1,I+1) = -D(1,I+1)
      GO TO 40
80    KOUNT = KOUNT + 1
      IF(D(2,I).LT.0.0)GO TO 90
      D(2,I) = -D(2,I)
90    IF(D(1,I).GT.0.0)GO TO 40
      GO TO(50,60,70,40)KOUNT
C
C    *** TEST THE RIGHT END FOUR ELEMENTS OF D ARRAY ***
C
100   IF(K.EQ.0)GO TO 101
      IF((ABS(D(1,I))-D(2,I)).GE.EPSLON)GO TO 130
      KOUNT = KOUNT + 1
      GO TO 105
101   IF((ABS(D(1,I))-D(2,I)).GT.EPSLON)GO TO 130
      KOUNT = KOUNT + 1
105   IF(D(1,I).LT.0.0)GO TO 110
      D(1,I) = -D(1,I)
      GO TO 150
110   IF(K.EQ.0)GO TO 111
      IF((ABS(D(2,I-1))-ABS(D(2,I))).GE.EPSLON)GO TO 130
      KOUNT = KOUNT + 1
      GO TO 115
111   IF((ABS(D(2,I-1))-ABS(D(2,I))).GT.EPSLON)GO TO 130
      KOUNT = KOUNT + 1
115   IF(D(2,I-1).LT.0.0)GO TO 120
      D(2,I-1) = -D(2,I-1)
      GO TO 150
120   IF(K.EQ.0)GO TO 121
      IF((ABS(D(1,I-1))-ABS(D(2,I))).GE.EPSLON)GO TO 130
      GO TO 125
121   IF((ABS(D(1,I-1))-ABS(D(2,I))).GT.EPSLON)GO TO 130
125   IF(D(1,I-1).LT.0.0)GO TO 150
      D(1,I-1) = -D(1,I-1)
      GO TO 150
130   KOUNT = KOUNT + 1
      IF(D(2,I).LT.0.0)GO TO 140
      D(2,I) = -D(2,I)
140   IF(D(1,I).GT.0.0.OR.D(2,I-1).GT.0.0)GO TO 150
      GO TO(110,120,150)KOUNT
150   CONTINUE
D     WRITE (6,*) ' '
D     WRITE (6,*) 'D ROWS',ROWKNT-1,' AND',ROWKNT,' :'
D     WRITE (6,5)
D5    FORMAT(1X,28('-'))
D     WRITE (6,3)(D(1,I),I=1,DCOL)
D     WRITE (6,*) ' '
D     WRITE (6,3)(D(2,I),I=1,DCOL)
D     WRITE (6,*) ' '
      RETURN

```


END

```

SUBROUTINE DRWCHG
C
C THIS SUBROUTINE REPLACES THE FIRST ROW OF ARRAY D WITH
C THE SECOND ROW.
C
C INCLUDE 'COMMON.FTN'
COMMON /DSTOR/ DCOL,X,D(2,M)
INTEGER DCOL
D WRITE (6,*) '*** ENTER DRWCHG ***'
C
C *** PERFORM THE ROW CHANGE OPERATION ***
C
C DO 10 I=1,DCOL
C D(1,I) = D(2,I)
10 CONTINUE
C
C RETURN
C END

```

AD-A127 409

DISPARITY ANALYSIS OF TIME-VARYING IMAGERY(U) AIR FORCE
INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF
ENGINEERING F D COOPER DEC 81 AFIT/GEO/EE/81D-1

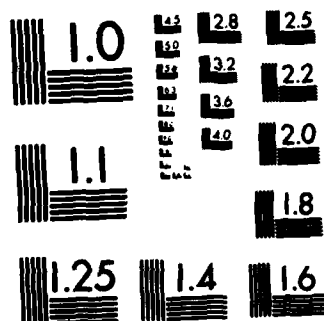
44

UNCLASSIFIED

F/G 20/6

NL

END
DATE
FILMED
5-83
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```

SUBROUTINE DVSTOR(DVAR)
C
C THIS SUBROUTINE STORES THE VALUES OF THE DIRECTED
C VARIANCES AS THEY ARE CALCULATED IN SUBROUTINE DVAR.
C
  INCLUDE 'COMMON.FTN'
  COMMON /DSTOR/ X,XX,D(2,M)
  COMMON /MOVE/ ROWKNT,COLKNT
  INTEGER ROWKNT,COLKNT,X
D  WRITE (6,*) '*** ENTER DVSTOR ***'
  IF(ROWKNT.EQ.1)GO TO 10
C
C *** STORE VALUES AFTER FIRST IMAGE ROW ***
C
  D(2,COLKNT) = DVAR
C
  RETURN
C
C *** STORE VALUES FOR FIRST IMAGE ROW ***
C
  D(1,COLKNT) = DVAR
C
  RETURN
  END

```

```

SUBROUTINE ADDDIF(INT)
C
C   THIS SUBROUTINE SUMS THE SQUARES OF THE DIFFERENCES OF
C   ADJACENT ELEMENTS IN FOUR DIRECTIONS FOR A COLUMN THE
C   WIDTH OF THE WINDOW FOR EACH COLUMN OF THE IMAGE.  THESE
C   SUMS ARE STORED IN ARRAYS X(*), Y(*), A(*), AND B(*).
C
  INCLUDE 'COMMON.FTN'
  VIRTUAL INT(2*N,M)
  COMMON /REG2/ X(M),Y(M),A(M),B(M)
  COMMON /WINDOW/ WNDROW
  COMMON /IMAGE/ XX,IMGCOL
  INTEGER WNDROW,XX
D  WRITE (6,*) '*** ENTER ADDDIF ***'
  DO 10 J=1,IMGCOL-1
    X(J) = 0
    Y(J) = 0
    A(J) = 0
    B(J) = 0
    DO 10 I=1,WNDROW-1
C
C   *** COMPUTE ADJACENT ELEMENT DIFFERENCES IN FOUR DIRECTIONS ***
C
    DIFROW = INT(I,J) - INT(I,J+1)
    DIFCOL = INT(I,J) - INT(I+1,J)
    DIFULD = INT(I,J) - INT(I+1,J+1)
    DIPLUD = INT(I+1,J) - INT(I,J+1)
C
C   *** COMPUTE SQUARES OF FOUR ADJACENT DIFFERENCES ***
C
    DFSQRW = DIFROW*DIFROW
    DFSQCL = DIFCOL*DIFCOL
    DFSQUL = DIFULD*DIFULD
    DFSQLU = DIPLUD*DIPLUD
C
C   *** FORM SUMS OF DIFFERENCE SQUARES FOR EACH COLUMN ***
C
    X(J) = X(J) + DFSQRW
    Y(J) = Y(J) + DFSQCL
    A(J) = A(J) + DFSQUL
    B(J) = B(J) + DFSQLU
C
C 10  CONTINUE
C
C   *** DETERMINE VALUES OF END SUMS ***
C
    Y(IMGCOL) = 0
    DO 20 I=1,WNDROW-1
      DFCLED = INT(I,IMGCOL) - INT(I+1,IMGCOL)

```

DFSQCE = DFCLED*DFCLED
Y(IMGCOL) = Y(IMGCOL) + DFSQCE
20 CONTINUE
X(IMGCOL) = X(IMGCOL-1)
A(IMGCOL) = A(IMGCOL-1)
B(IMGCOL) = B(IMGCOL-1)
C
RETURN
END

```

SUBROUTINE EPSFND(EPSLON)
C
C THIS SUBROUTINE ALLOWS THE USER TO INPUT AN EPSILON
C VALUE USED TO TEST FOR INTERESTING POINTS.
C
1  TYPE *, ' '
   TYPE *, 'WHAT VALUE OF EPSILON DO YOU WISH TO USE?'
   TYPE *, ' '
   TYPE 2
2  FORMAT(5X, 'EPSILON = ', $)
   ACCEPT *, EPSLON
   TYPE *, ' '
   TYPE *, 'HERE IS WHAT YOU HAVE INPUT:'
   TYPE *, ' '
C
C *** DISPLAY THE VALUE OF EPSILON ***
C
   TYPE *, 'EPSILON = ', EPSLON
C
C *** VERIFY THE CORRECT RESPONSE ***
C
   CALL VERIFY(ANSWER)
C
   IF(ANSWER.EQ.'NO')GO TO 1
   WRITE (6,*) ' '
   WRITE (6,*) '*** EPSILON VALUE FOR DIRECTED VARIANCE TESTS ***'
   WRITE (6,*) ' '
   WRITE (6,*) 'EPSILON = ', EPSLON
   WRITE (6,*) ' '
   RETURN
   END

```



```

SUBROUTINE IOP3(INT,IOPERR)
C
C   THIS SUBROUTINE COMPUTES THE INTERESTING POINTS OF
C   THE SELECTED IMAGE USING THE METHOD OF EDGED VARIANCE.
C   THE DIRECTED VARIANCE IN FOUR DIRECTIONS IS FIRST
C   COMPUTED USING SUBROUTINE DVAR. THE MINIMUM OF FOUR
C   PERPENDICULAR RATIOS IS THEN FOUND. NEXT, THE SIMPLE
C   VARIANCE IS COMPUTED USING SUBROUTINE SVAR. THE EDGED
C   VARIANCE IS FORMED BY THE PRODUCT OF THE SIMPLE
C   VARIANCE AND THE MINIMUM OF THE FOUR PERPENDICULAR
C   RATIOS. EACH COMPUTED VALUE OF EDGED VARIANCE IS
C   TESTED AGAINST A USER-DEFINED THRESHOLD. IF THE VALUE
C   OF THE EDGED VARIANCE IS GREATER THAN THE THRESHOLD
C   VALUE, THAT POINT IS DEFINED TO BE AN INTERESTING POINT.
C
D   WRITE (6,*) '*** ENTER IOP3 ***'
C
C   *** SET THRESHOLD VALUE ***
C
C   CALI. THRHLD(THRESH)
C
C   *** INITIALIZE INTEREST OPERATOR PARAMETERS ***
C
C   CALL INITAL(IMGEND,IPSEND)
C
C   *** COMPUTE THE EDGED VARIANCE FOR EACH WINDOW POSITION ***
C
10  CALL EVAR(INT,EDGVAR,IOPERR)
   IF(IOPERR.EQ.1)GO TO 20
C
C   *** TEST COMPUTED VARIANCE TO SEE IF INTERESTING POINT EXISTS ***
C
   IF(EDGVAR.GT.THRESH)CALL IPSTOR(EDGVAR,IOPERR,IPSEND)
   IF(IOPERR.EQ.1.OR.IPSEND.EQ.1)GO TO 20
C
C   *** MOVE WINDOW ONE IMAGE ELEMENT ***
C
   CALL WNDMOV(IMGEND)
   IF(IMGEND.NE.1)GO TO 10
C
20  CALL CLOSKP(IOPERR)
   RETURN
   END

```

```

C      SUBROUTINE EVAR(INT,EDGVAR,VARERR)
C
C      THIS SUBROUTINE COMPUTES THE VALUE OF EDGED VARIANCE FROM
C      THE SIMPLE VARIANCE CALCULATED IN SUBROUTINE SVAR, AND THE
C      FOUR VALUES OF DIRECTED VARIANCE CALCULATED IN SUBROUTINE
C      DVAR.
C
C      COMMON /EDVAR/ EFLAG,RTOMIN
C      INTEGER VARERR,EFLAG
D      WRITE (6,*) '*** ENTER EVAR ***'
C      EFLAG = 1
C
C      *** COMPUTE THE SIMPLE VARIANCE ***
C
C      CALL SVAR(INT,VARNCE,VARERR)
C      IF(VARERR.EQ.1)RETURN
C
C      *** COMPUTE THE MINIMUM OF THE FOUR PERPENDICULAR RATIOS
C      OF DIRECTED VARIANCE ***
C
C      CALL DVAR(INT,VARERR)
C
C      *** COMPUTE THE EDGED VARIANCE ***
C
C      EDGVAR = VARNCE*RTOMIN
C
D      WRITE (6,*) 'EDGVAR = ',EDGVAR
D      WRITE (6,*) ' '
C      RETURN
C      END

```

```

SUBROUTINE RATIO(DVAR1,DVAR2,DVAR3,DVAR4,RATIO1,RATIO2,RATIO3,
1RATIO4)
C
C THIS SUBROUTINE COMPUTES FOUR PERPENDICULAR RATIOS OF THE FOUR
C DIRECTED VARIANCE VALUES CALCULATED IN SUBROUTINE DVAR. IF THE
C DENOMINATOR OF ONE OF THE RATIOS IS ZERO, AND THE NUMERATOR IS
C NONZERO, THAT RATIO IS SET TO A BIG NUMBER. IF BOTH NUMERATOR
C AND DENOMINATOR ARE ZERO, THE RATIOS IN BOTH DIRECTIONS ARE SET
C TO ZERO.
C
PARAMETER BIGNBR = 1.0E+35
IF(DVAR2.EQ.0.0)GO TO 10
RATIO1 = DVAR1/DVAR2
IF(DVAR1.EQ.0.0)GO TO 20
RATIO2 = DVAR2/DVAR1
5 IF(DVAR4.EQ.0.0)GO TO 30
RATIO3 = DVAR3/DVAR4
IF(DVAR3.EQ.0.0)GO TO 40
RATIO4 = DVAR4/DVAR3
GO TO 100
10 IF(DVAR1.EQ.0.0)GO TO 50
RATIO1 = BIGNBR
RATIO2 = 0.0
GO TO 5
20 RATIO2 = BIGNBR
GO TO 5
30 IF(DVAR3.EQ.0.0)GO TO 60
RATIO3 = BIGNBR
RATIO4 = 0.0
GO TO 100
40 RATIO4 = BIGNBR
GO TO 100
50 RATIO1 = 0.0
RATIO2 = 0.0
GO TO 5
60 RATIO3 = 0.0
RATIO4 = 0.0
100 CONTINUE
D WRITE (6,101) RATIO1,RATIO2,RATIO3,RATIO4
D101 FORMAT(1X,'RATIO1 = ',G15.7,2X,'RATIO2 = ',G15.7,2X,'RATIO3 = '
D 1,G15.7,2X,'RATIO4 = ',G15.7,/)
RETURN
END

```

SUBROUTINE IOP4(IOPERR)
TYPE *, 'IOP4 NOT AVAILABLE AT THIS TIME'
RETURN
END

SUBROUTINE IOP5(IOPERR)
TYPE *, 'IOP5 NOT AVAILABLE AT THIS TIME'
RETURN
END

SUBROUTINE IOP6(IOPERR)
TYPE *, 'IOP6 NOT AVAILABLE AT THIS TIME'
RETURN
END

SUBROUTINE IOP7(IOPERR)
TYPE *, 'IOP7 NOT AVAILABLE AT THIS TIME'
RETURN
END

SUBROUTINE IOP8(IOPERR)
TYPE *, 'IOP8 NOT AVAILABLE AT THIS TIME'
RETURN
END

SUBROUTINE IOP9(IOPERR)
TYPE *, 'IOP9 NOT AVAILABLE AT THIS TIME'
RETURN
END

SUBROUTINE IOP10(IOPERR)
TYPE *, 'IOP10 NOT AVAILABLE AT THIS TIME'
RETURN
END

```

SUBROUTINE IPSTOR(VALUE,IPSERR,IPSEND)

C
C THIS SUBROUTINE STORES THE COMPUTED INTERESTING POINTS
C IN THREE ARRAYS. ARRAY IPR(*) CONTAINS THE ROW NUMBER
C OF THE INTERESTING POINTS. ARRAY IPC(*) CONTAINS THE
C COLUMN NUMBER OF THE INTERESTING POINTS. ARRAY VALUIP(*)
C CONTAINS THE VALUE OF THE VARIANCE AT THE INTERESTING
C POINT LOCATION. A CORRELATION WINDOW IS ALSO STORED AROUND
C EACH INTERESTING POINT IF MATCHING IS DESIRED (IPFLAG=1).
C

INCLUDE 'COMMON.FTN'
COMMON /IOP/ IX,IPFLAG
COMMON /MOVE/ ROWKNT,COLKNT
COMMON /WINDOW/ X,XX,WNDRWH,WNDCLH
COMMON /IPS/ IPKNT,IPR(M),IPC(M),VALUIP(M)
INTEGER ROWKNT,COLKNT,X,XX,WNDRWH,WNDCLH
WRITE (6,*) '*** ENTER IPSTOR ***'
IPKNT = IPKNT + 1

C
C *** DETERMINE EACH INTERESTING POINT LOCATION ***
C
IPROW = ROWKNT + WNDRWH
IPCOL = COLKNT + WNDCLH

C
C *** STORE THE INTERESTING POINT INFORMATION ***
C
IPR(IPKNT) = IPROW
IPC(IPKNT) = IPCOL
VALUIP(IPKNT) = VALUE

C
IF(IPFLAG.EQ.0)GO TO 5

C
C *** FORM INTERESTING POINT CORRELATION WINDOW ***
C
CALL CWDOUT(IPROW,IPCOL,IPSERR)
IF(IPSERR.EQ.1)RETURN

C
5 IF(IPKNT.GE.M)GO TO 40
RETURN
40 WRITE (6,*)' '
WRITE (6,*)'*** INTERESTING POINT STORAGE LIMIT EXCEEDED ***'
WRITE (6,*)' '
WRITE (6,*)'*****'
1***'
WRITE (6,*)'*** WARNING: ENTIRE IMAGE MAY NOT HAVE BEEN PROCESSED'
1***'
WRITE (6,*)'*****'
1***'
WRITE (6,*)' '

```

```
WRITE (6,*) 'PROCESSING TERMINATED AT WINDOW POSITION (CENTER):'  
WRITE (6,*) ' '  
WRITE (6,*) '      ROW NUMBER = ', IPROW  
WRITE (6,*) '      COLUMN NUMBER = ', IPCOL  
WRITE (6,*) ' '  
TYPE *, ' '  
TYPE *, '*** INTERESTING POINT STORAGE LIMIT EXCEEDED ***'  
IPSEND = 1  
RETURN  
END
```

```

SUBROUTINE CWDOUT(IPROW,IPCOL,CWDERR)

C
C THIS SUBROUTINE IS USED TO STORE THE CORRELATION WINDOWS
C SURROUNDING EACH INTERESTING POINT ON A TEMPORARY SCRATCH
C FILE.
C
  INCLUDE 'COMMON.FTN'
  COMMON /CWFILE/ LUNCW
  COMMON /CORWND/ IW,IX,IY,IZ,CWDSZE
  COMMON /IPS/ IPKNT,IPR(M),IPC(M),X(M),IPWND(M/3)
  INTEGER CWDSZE,CWDERR

C
C *** OBTAIN THE CORRELATION WINDOW FOR EACH INTERESTING POINT ***
C
  CALL CORWND(IPROW,IPCOL,CWDERR)
  IF(CWDERR.EQ.1)RETURN

C
C *** OUTPUT EACH CORRELATION WINDOW ***
C
  WRITE(LUNCW,10,ERR=100)(IPWND(I),I=1,CWDSZE)
10  FORMAT(<CWDSZE>I3)
C
  RETURN
100 TYPE *, ' '
  TYPE *, '*** WRITE ERROR OCCURED IN CWDOUT ON LOGICAL UNIT',LUNCW,
  1 ' ***'
  CLOSE(UNIT=LUNCW,DISPOSE='DELETE')
  CALL CLOSKP(CWDERR)
  CWDERR = 1
  WRITE (6,*) '*** EXIT CWDOUT - WRITE ERROR TERMINATION ***'
  RETURN
  END

```

```

SUBROUTINE CORWND(IPROW,IPCOL,CORERR)
C
C THIS SUBROUTINE FORMS A CORRELATION WINDOW AROUND EACH INTEREST-
C ING POINT OF A SIZE DETERMINED BY THE USER IN SUBROUTINE CWDSE.
C IF THE CORRELATION WINDOW EXTENDS BEYOND THE EDGES OF THE IMAGE
C FOR POINTS CLOSE TO THE EDGE, THE PIXELS OUTSIDE THE IMAGE AREA
C BECOME "PSEUDO-PIXELS", AND TAKE ON THE VALUE OF THEIR CORRES-
C PONDING PIXEL ON THE IMAGE EDGE.
C
C INCLUDE 'COMMON.FTN'
C COMMON /IBUF/ IBUF(M)
C COMMON /INBUF/ INTBUF(M)
C COMMON /IMAGE/ IMGROW,IMGCOL
C COMMON /CORWND/ CWDROW,CWDCOL,CWDRWH,CWDCLH
C COMMON /IPS/ IX,IPR(M),IPC(M),X(M),IPCWND(M/3)
C INTEGER CWDROW,CWDCOL,CWDRWH,CWDCLH,CORERR
C INTEGER CWDROWS,CWDCLS,CWDRWE,CWDCLE,RECNUM
C LOGICAL*1 IBUF
C WRITE (6,*) '*** ENTER CORWND ***'
C
C *** FIND CORRELATION WINDOW START AND END LOCATIONS ***
C
C CALL CWDTST(IPROW,IPCOL,CWDROWS,CWDCLS,CWDRWE,CWDCLE)
C
C RECNUM = IPROW - CWDRWH + CWDROWS
C IPCKNT = CWDCOL*(CWDROWS-1) + 1
C IMCLST = IPCOL - CWDCLH
C
C *** INPUT IMAGE INTENSITY VALUES ***
C
C CALL INBUFR(RECNUM,CORERR)
C IF(CORERR.EQ.1)RETURN
C
C *** PERFORM BYTE TO WORD NUMBER CONVERSION ***
C
C DO 2 J=1,IMGCOL
C INTBUF(J) = IBUF(J)
C IF(INTBUF(J).LT.0)INTBUF(J) = INTBUF(J) + 256
C
C 2 CONTINUE
C
C IF(CWDCLS.GT.1)GO TO 10
C
C *** FORM MIDDLE WINDOW ROW ELEMENTS ***
C
C DO 4 K=CWDCLS,CWDCLE
C IPCWND(IPCKNT) = INTBUF(IMCLST+K)
C IPCKNT = IPCKNT + 1
C
C 4 CONTINUE
C

```



```

      IF(CWDCLE.LT.CWDCOL)GO TO 20
5      IF(CWDRWS.GT.1)GO TO 30
6      IF(CWDRWE.GT.1.AND.RECNUM.EQ.IMGROW)GO TO 40
      RECNUM = RECNUM + 1
      IF(RECNUM.EQ.IPROW+CWDRWH)GO TO 50
      GO TO 1

C
C      *** FORM LEFT WINDOW ROW ELEMENTS ***
C
10     DO 11 K=1,CWDCLS-1
      IPCWND(IPCKNT) = INTBUF(1)
      IPCKNT = IPCKNT + 1
11     CONTINUE
      GO TO 3

C
C      *** FORM RIGHT WINDOW ROW ELEMENTS ***
C
20     DO 21 K=CWDCLE+1,CWDCOL
      IPCWND(IPCKNT) = INTBUF(IMGCOL)
      IPCKNT = IPCKNT + 1
21     CONTINUE
      GO TO 5

C
C      *** FORM WINDOW ROWS ABOVE IMAGE ***
C
30     IPCKNT = 1
      IMRWST = CWDCOL*(CWDRWS-1)
      DO 31 J=1,CWDRWS-1
      DO 31 K=1,CWDCOL
      IPCWND(IPCKNT) = IPCWND(IMRWST+K)
      IPCKNT = IPCKNT + 1
31     CONTINUE
      IPCKNT = CWDCOL*CWDRWS + 1
      CWDRWS = 1
      GO TO 6

C
C      *** FORM WINDOW ROWS BELOW IMAGE ***
C
40     IMRWED = CWDCOL*(CWDRWE-1)
      DO 41 J=CWDRWE+1,CWDROW
      DO 41 K=1,CWDCOL
      IPCWND(IPCKNT) = IPCWND(IMRWED+K)
      IPCKNT = IPCKNT + 1
41     CONTINUE
50     CONTINUE

C
C      *** PRINT CORRELATION WINDOW ***
C
D      IPCSIZE = CWDROW*CWDCOL
D      WRITE (6,*) ' '

```

```

D      WRITE (6,*) 'CORRELATION WINDOW/ FOR INTERESTING POINT:',IPROW,
D      1',',IPCOL
D      WRITE (6,51)
D51    FORMAT(1X,57('-'))
D      WRITE (6,52)(IPCWND(I),I=1,IPCSZE)
D52    FORMAT(<CWDCOL>(2X,I3))
D      WRITE (6,*) ' '
C
      RETURN
      END

```

```

SUBROUTINE CWDTST(IPROW,IPCOL,CWDRWS,CWDCLS,CWDRWE,CWDCLE)

C
C THIS SUBROUTINE TESTS THE POSITION OF THE CORRELATION WINDOW WITH
C RESPECT TO THE IMAGE TO DETERMINE IF THE IMAGE BOUNDARY HAS BEEN
C EXCEEDED. IF SO, THE CORRELATION WINDOW ROW AND COLUMN START AND
C STOP LOCATIONS ARE DETERMINED WITH RESPECT TO THE IMAGE.
C

COMMON /IMAGE/ IMGROW,IMGCOL
COMMON /CORWND/ CWDROW,CWDCOL,CWDRWH,CWDCLH
INTEGER CWDROW,CWDCOL,CWDRWH,CWDCLH
INTEGER CWDLCL,CWDTRW,CWDRCL,CWDBRW
INTEGER CWDRWS,CWDRWE,CWDCLS,CWDCLE
WRITE (6,*) '*** ENTER CWDTST ***'

D
C *** TEST FOR LEFT COLUMN OF IMAGE ***
C
1 CWDLCL = IPCOL - CWDCLH
  IF(CWDLCL.LT.0)GO TO 10
  CWDCLS = 1

C
C *** TEST FOR TOP OF IMAGE ***
C
2 CWDTRW = IPROW - CWDRWH
  IF(CWDTRW.LT.0)GO TO 20
  CWDRWS = 1

C
C *** TEST FOR RIGHT COLUMN OF IMAGE ***
C
3 CWDRCL = IMGCOL - (IPCOL+CWDCLH) + 1
  IF(CWDRCL.LT.0)GO TO 30
  CWDCLE = CWDCOL

C
C *** TEST FOR BOTTOM OF IMAGE ***
C
4 CWDBRW = IMGROW - (IPROW+CWDRWH) + 1
  IF(CWDBRW.LT.0)GO TO 40
  CWDRWE = 1

C
  RETURN

C
C *** DETERMINE COLUMN START POSITION ***
C
10 CWDCLS = 1 - CWDLCL
   GO TO 2

C
C *** DETERMINE ROW START POSITION ***
C
20 CWDRWS = 1 - CWDTRW

```

```

GO TO 3
C
C   *** DETERMINE COLUMN END POSITION ***
C
30  CWDCL = CWDCL + CWDCL
    GO TO 4
C
C   *** DETERMINE ROW END POSITION ***
C
40  CWDRE = CWDROW + CWDBRW
C
RETURN
END

```

```

SUBROUTINE INBUFR(RECNUM, INBERR)
C
C THIS SUBROUTINE INPUTS ONE RECORD FROM THE FILE ASSIGNED TO
C LOGICAL UNIT NUMBER LUN AND STORES IT IN ARRAY IBUF.
C
INCLUDE 'COMMON.FTN'
COMMON /IBUF/ IBUF(M)
COMMON /FILE/ LUN
COMMON /IMAGE/ IX, IMGCOL
LOGICAL*1 IBUF
INTEGER RECNUM
C
C *** READ THE FILE RECORD ***
C
READ(LUN, RECNUM, END=10, ERR=100)(IBUF(I), I=1, IMGCOL)
C
10 CONTINUE
RETURN
100 TYPE *, ' '
TYPE *, '*** READ ERROR OCCURED IN INBUFR ON LOGICAL UNIT', LUN,
1 ' ***'
CALL CLOSKP(INBERR)
INBERR = 1
WRITE (6,*) '*** EXIT INBUFR - READ ERROR TERMINATION ***'
RETURN
END

```

SUBROUTINE IPLST

```
C
C THIS SUBROUTINE LISTS THE ROW AND COLUMN POSITIONS AND VALUE
C OF EACH INTERESTING POINT AFTER INTERESTING POINT PROCESSING
C IS COMPLETE.
C
C INCLUDE 'COMMON.FTN'
C COMMON /IPS/ IPKNT,IPR(M),IPC(M),VALUIP(M)
C
C *** LIST THE INTERESTING POINTS ***
C
C TYPE *, ' '
C TYPE *, '*** INTERESTING POINT PROCESSING COMPLETE ***'
10 WRITE (6,*) ' '
C WRITE (6,*) 'HERE IS THE LIST OF INTERESTING POINT LOCATIONS:'
C WRITE (6,*) '-----'
C DO 30 I=1,IPKNT
C WRITE (6,20) I,IPR(I),IPC(I),I,VALUIP(I)
20 FORMAT(3X,'IP(',I3,') = ',I3,',',I3,5X,'IPMAG(',I3,') = ',F10.3)
30 CONTINUE
C WRITE (6,*) ' '
C WRITE (6,*) ' '
C RETURN
C END
```

Appendix E: Interesting Point Matching Section

This appendix contains the FORTRAN source listing for the Interesting Point Matching Section of program DIDA.

```

SUBROUTINE IPMTCH(STORE,IPMERR,IPMEND)
C
C   THIS SUBROUTINE ALLOWS THE USER TO OPEN THE FILES AND INPUT THE
C   TWO SETS OF INTERESTING POINTS FROM IMAGE 1 AND IMAGE 2 TO BE
C   MATCHED. THE MATCHING ALGORITHM IS THEN SELECTED AND THE MATCH-
C   ING PROCESS BEGINS. WHEN COMPLETED, THE MATCHED POINTS ARE
C   LISTED ON THE PRINTER.
C
COMMON /IFILE1/ LUN1
COMMON /IFILE2/ LUN2
D   WRITE (6,*) '*** ENTER IPMTCH ***'
C
C   *** OPEN THE FILES OF THE TWO SETS OF INTERESTING POINTS TO
C   BE INPUT ***
C
CALL IPSOPN(IPMERR,IPMEND)
IF(IPMERR.EQ.1.OR.IPMEND.EQ.1)RETURN
C
C   *** SELECT THE MATCHING ALGORITHM ***
C
CALL MTASEL(STORE,IPMERR,IPMEND)
IF(IPMERR.EQ.1.OR.IPMEND.EQ.1)RETURN
C
C   *** LIST THE MATCHED POINTS ***
C
CALL MPLST(STORE)
C
CLOSE(UNIT=LUN1,DISPOSE='KEEP')
CLOSE(UNIT=LUN2,DISPOSE='KEEP')
RETURN
END
```

```

SUBROUTINE IPSOPN(IPSERR,IPSEND)

C
C THIS SUBROUTINE OPENS THE FILES OF THE TWO SETS OF INTERESTING
C POINTS TO BE INPUT FOR MATCHING. THE FIRST RECORD OF EACH FILE
C IS THEN INPUT AND STORED IN A COMMON AREA. A COMPARISON OF THE
C TWO SETS IS THEN PERFORMED TO ALLOW THE USER TO DECIDE IF THE
C MATCHING PROCESS SHOULD CONTINUE.
C
COMMON /IFILE1/ LUN1
COMMON /IFILE2/ LUN2
COMMON /IPS1/ IPKNT1,IPTYP1,CWROW1,CWCOL1,CWSZE1
COMMON /IPS2/ IPKNT2,IPTYP2,CWROW2,CWCOL2,CWSZE2
INTEGER CWROW1,CWCOL1,CWSZE1,CWROW2,CWCOL2,CWSZE2
D WRITE (6,*) '*** ENTER IPSOPN ***'
TYPE *, ' '
TYPE *, 'ENTER THE LOGICAL UNIT NUMBER AND FILE NAME YOU WISH TO'
TYPE *, 'USE TO INPUT THE FIRST SET OF INTERESTING POINTS:'
C
C *** ENTER THE LOGICAL UNIT NUMBER AND FILE NAME ***
C
CALL IF1FLU
C
C *** OPEN THE FILE FOR INTERESTING POINT SET 1 ***
C
CALL OPNIF1(IPSERR)
IF(IPSERR.EQ.1)RETURN
C
C *** READ THE FIRST RECORD ***
C
READ(LUN1,1,END=10,ERR=100) IPKNT1,IPTYP1,CWROW1,CWCOL1,CWSZE1
1 FORMAT(5I3)
C
10 TYPE *, ' '
TYPE *, 'ENTER THE LOGICAL UNIT NUMBER AND FILE NAME YOU WISH TO'
TYPE *, 'USE TO INPUT THE SECOND SET OF INTERESTING POINTS:'
C
C *** ENTER THE LOGICAL UNIT NUMBER AND FILE NAME ***
C
CALL IF2FLU
C
C *** OPEN THE FILE FOR INTERESTING POINT SET 2 ***
C
CALL OPNIF2(IPSERR)
IF(IPSERR.EQ.1)RETURN
C
C *** READ THE FIRST RECORD ***
C
READ(LUN2,1,END=20,ERR=200) IPKNT2,IPTYP2,CWROW2,CWCOL2,CWSZE2
C

```



```

20      REWIND LUN2
        TYPE *, ' '
        TYPE *, 'HERE IS THE COMPARISON BETWEEN INTERESTING POINT SETS'
        TYPE *, 'ONE AND TWO:'
        TYPE *, ' '
        TYPE *, '          IP SET 1          IP SET 2'
        TYPE *, '-----'
        TYPE *, 'NUMBER OF POINTS = ', IPKNT1, ' NUMBER OF POINTS = ', IPKNT2
        TYPE *, 'TYPE INTEREST OP = ', IPTYP1, ' TYPE INTEREST OP = ', IPTYP2
        TYPE *, 'COR WINDOW ROWS = ', CWROW1, ' COR WINDOW ROWS = ', CWROW2
        TYPE *, 'COR WINDOW COLS = ', CWCOL1, ' COR WINDOW COLS = ', CWCOL2
        TYPE *, 'COR WINDOW SIZE = ', CWSZE1, ' COR WINDOW SIZE = ', CWSZE2
        TYPE *, ' '
        TYPE *, 'NOTE: IT IS NOT NECESSARY FOR THE CORRELATION WINDOW ROW,'
        TYPE *, 'COLUMN, OR OVERALL SIZES TO MATCH TO PERFORM MATCHING!'
        TYPE *, ' '
        TYPE *, 'DO YOU WISH TO CONTINUE WITH THE MATCHING PROCESS?'
        TYPE *, 'IF YOU DO, TYPE YES. OTHERWISE, TYPE NO.'
        TYPE *, ' '
        TYPE 30
30      FORMAT(5X, 'ANSWER = ', S)
        ACCEPT 40, ANSWER
40      FORMAT(A3)
        CALL ANSCHK(ANSWER)
        IF(ANSWER.EQ.'YES')RETURN
        CLOSE(UNIT=LUN1, DISPOSE='KEEP')
        CLOSE(UNIT=LUN2, DISPOSE='KEEP')
        IPSEND = 1
        RETURN
100     TYPE *, ' '
        TYPE *, '*** READ ERROR OCCURED IN IPSOPN ON LOGICAL UNIT', LUN1,
        1' ***'
        CLOSE(UNIT=LUN1, DISPOSE='KEEP')
        IPSERR = 1
        WRITE (6,*) '*** EXIT IPSOPN - READ ERROR TERMINATION ***'
        RETURN
200     TYPE *, ' '
        TYPE *, '*** READ ERROR OCCURED IN IPSOPN ON LOGICAL UNIT', LUN2,
        1' ***'
        CLOSE(UNIT=LUN1, DISPOSE='KEEP')
        CLOSE(UNIT=LUN2, DISPOSE='KEEP')
        IPSERR = 1
        WRITE (6,*) '*** EXIT IPSOPN - READ ERROR TERMINATION ***'
        RETURN
        END

```

```

C      SUBROUTINE IF2FLU
C
C      THIS SUBROUTINE ALLOWS THE USER TO ENTER THE LOGICAL UNIT
C      NUMBER AND FILE NAME INTERACTIVELY FOR INTERESTING POINT
C      SET INPUT/OUTPUT OPERATIONS.  THE USER MAY CORRECT THIS
C      INFORMATION IF AN ENTRY ERROR OCCURS.
C
C      COMMON /IFILE2/ LUN2,NCHAR2,FNAME2(34)
C      LOGICAL*1 FNAME2
C
C      *** ENTER THE LOGICAL UNIT NUMBER AND FILE NAME ***
C
C      CALL LUNBR2
C      CALL FILNM2
C
C      *** OPTIONAL ENTRY INFORMATION CHECK ***
C
C      TYPE *, ' '
C      TYPE *, 'DO YOU WISH TO CHECK FOR ENTRY ERRORS?'
C      TYPE *, 'IF YOU DO, TYPE YES.  OTHERWISE, TYPE NO.'
C      TYPE *, ' '
C      TYPE 1
1      FORMAT(5X, 'ANSWER = ', S)
C      ACCEPT 2, ANSWER
2      FORMAT(A3)
C
C      *** CHECK ANSWER ENTRY ***
C
C      CALL ANSCHK(ANSWER)
C
C      IF(ANSWER.EQ.'NO')GO TO 3
C
C      *** CHECK ENTRY INFORMATION ***
C
C      CALL IF2CHK
C
3      WRITE (6,*) ' '
C      WRITE (6,*) '*** INTERESTING POINT SET 2 FILE INFORMATION ***'
C      WRITE (6,*) ' '
C      WRITE (6,*) 'LOGICAL UNIT NUMBER = ', LUN2
C      WRITE (6,4) (FNAME2(I), I=1, NCHAR2)
4      FORMAT(1X, 'FILENAME = ', <NCHAR2+1>A1)
C      WRITE (6,*) ' '
C      RETURN
C      END

```

```

SUBROUTINE IF2CHK
C
C THIS SUBROUTINE ALLOWS THE USER TO CHECK THE INFORMATION
C INPUT FOR THE FILE NAME AND LOGICAL UNIT NUMBER, AND TO
C CORRECT THEM IF DESIRED.
C
COMMON /IFILE2/ LUN2,NCHAR2,FNAME2(34)
LOGICAL*1 FNAME2
1 TYPE *,' '
TYPE *,'HERE IS WHAT YOU HAVE INPUT:'
TYPE *,' '
C
C *** DISPLAY THE FILE NAME AND LOGICAL UNIT NUMBER INPUT ***
C
TYPE *,'LOGICAL UNIT NUMBER = ',LUN2
TYPE 2,(FNAME2(I),I=1,NCHAR2)
2 FORMAT(1X,'FILENAME = ',<NCHAR2+1>A1)
C
C *** VERIFY THE CORRECT RESPONSE ***
C
CALL VERIFY(ANSWER)
C
IF(ANSWER.EQ.'YES')GO TO 3
C
C *** CHANGE INCORRECT INFORMATION ***
C
CALL IF2COR
C
GO TO 1
3 RETURN
END

```

```

SUBROUTINE IF2COR
C
C THIS SUBROUTINE ALLOWS THE USER TO CORRECT THE INFORMATION
C INPUT FOR THE FILE NAME AND LOGICAL UNIT NUMBER.
C
REAL ITEM
C
C *** MAKE ERROR CORRECTIONS ***
C
TYPE *, ' '
TYPE *, 'DO YOU WANT TO CHANGE THE LOGICAL UNIT NUMBER, '
TYPE *, 'FILE NAME, OR BOTH?'
1 TYPE *, 'IF YOU WANT TO CHANGE THE LOGICAL UNIT NUMBER, TYPE LUN.'
TYPE *, 'IF YOU WANT TO CHANGE THE FILE NAME, TYPE FILE.'
TYPE *, 'IF YOU WANT TO CHANGE BOTH, TYPE BOTH.'
TYPE *, ' '
TYPE 2
2 FORMAT(5X, 'ITEM YOU WISH TO CHANGE = ', $)
ACCEPT 3, ITEM
3 FORMAT(A4)
IF(ITEM.EQ.'BOTH')GO TO 10
IF(ITEM.EQ.'LUN')GO TO 20
IF(ITEM.EQ.'FILE')GO TO 30
TYPE *, ' '
TYPE *, '*** ITEM ENTRY ERROR - PLEASE RETYPE ***'
TYPE *, ' '
GO TO 1
C
C *** CHANGE BOTH FILE NAME AND LOGICAL UNIT NUMBER ***
C
10 CALL FILM2
CALL LUNBR2
RETURN
C
C *** CHANGE LOGICAL UNIT NUMBER ***
C
20 CALL LUNBR2
RETURN
C
C *** CHANGE FILE NAME ***
C
30 CALL FILM2
RETURN
END

```

```

SUBROUTINE FILNM2

C
C THIS SUBROUTINE ALLOWS THE USER TO INPUT THE FILE NAME OF
C THE FILE BEING USED FOR INTERESTING POINT SET INPUT/OUTPUT
C OPERATIONS.
C

COMMON /IFILE2/ IX,NCHAR2,FNAME2(34)
LOGICAL*1 FNAME2
TYPE *,' '
TYPE *,'WHAT DEVICE/UIC/FILENAME DO YOU WISH TO USE?'
TYPE *,'ENTER ACCORDING TO THE FOLLOWING FORMAT:'
TYPE *,' '
TYPE *,'          DEV:[ UIC ]FILENAME.TYPE;VERSION'
TYPE *,'EXAMPLE:  DK1:[10,20]FILENAME.IMG;3'
TYPE *,' '

C
C *** INPUT THE DESIRED FILE NAME ***
C

TYPE 1
FORMAT(5X,'FILENAME = ',S)
ACCEPT 2,NCHAR2,FNAME2
2
FORMAT(Q,34A1)
FNAME2(NCHAR2+1) = 0

C

RETURN
END

```

```

SUBROUTINE LUNBR2

C
C THIS SUBROUTINE ALLOWS THE USER TO INPUT THE LOGICAL UNIT
C NUMBER ASSOCIATED WITH THE FILE TO BE USED FOR INTERESTING
C POINT SET INPUT/OUTPUT OPERATIONS.
C

COMMON /IFILE1/ LUN1
COMMON /IFILE2/ LUN2
1  TYPE *, ' '
   TYPE *, 'WHAT LOGICAL UNIT NUMBER YOU WISH TO USE (1-10)?'
   TYPE *, ' '

C
C *** INPUT THE DESIRED LOGICAL UNIT NUMBER ***
C
   TYPE 2
2  FORMAT(5X, 'LOGICAL UNIT NUMBER = ', $)
   ACCEPT *, LUN2

C
C *** CHECK FOR INCORRECT NUMBER ***
C
   IF(LUN2.LT.1.OR.LUN2.GT.10)GO TO 3
   IF(LUN2.EQ.5)GO TO 4
   IF(LUN2.EQ.6)GO TO 5
   IF(LUN2.EQ.LUN1)GO TO 6

C
   RETURN
3  TYPE *, ' '
   TYPE *, '*** INVALID LOGICAL UNIT NUMBER', LUN2, ' ATTEMPTED ***'
   TYPE *, ' '
   TYPE *, 'LOGICAL UNIT NUMBER MUST BE BETWEEN 1 AND 10.'
   GO TO 1
4  TYPE *, ' '
   TYPE *, 'LOGICAL UNIT 5 IS RESERVED FOR THE USERS TERMINAL.'
   TYPE *, 'PLEASE USE ANOTHER NUMBER!'
   GO TO 1
5  TYPE *, ' '
   TYPE *, 'LOGICAL UNIT 6 IS RESERVED FOR THE LINE PRINTER.'
   TYPE *, 'PLEASE USE ANOTHER NUMBER!'
   GO TO 1
6  TYPE *, ' '
   TYPE *, 'LOGICAL UNIT NUMBER', LUN1, ' ALREADY IN USE.'
   TYPE *, 'PLEASE USE ANOTHER NUMBER!'
   GO TO 1
END

```

```

SUBROUTINE OPNIF2(OPNERR)
C
C   THIS SUBROUTINE OPENS A FILE FOR INPUTTING THE SECOND SET OF INT-
C   ERESTING POINTS FOR MATCHING.  THE FILE IS A SEQUENTIAL FORMATTED
C   FILE.
C
C   INCLUDE 'COMMON.FTN'
C   COMMON /IFILE2/ LUN2,IX,FNAME2(34)
C   LOGICAL*1 FNAME2
C   INTEGER OPNERR
C
C   *** OPEN THE FILE ***
C
C   OPEN(UNIT=LUN2,NAME=FNAME2,DISPOSE='KEEP',RECORDSIZE=M+20,
C   1   TYPE='OLD',ERR=100)
C
C   RETURN
100  TYPE *, ' '
    TYPE *, '*** OPEN ERROR OCCURED IN OPNIF2 ON LOGICAL UNIT',LUN2,
    1 ' ***'
    CLOSE(UNIT=LUN2,DISPOSE='KEEP',ERR=200)
    OPNERR = 1
    WRITE (6,*) '*** EXIT OPNIF2 - OPEN ERROR TERMINATION ***'
    RETURN
200  TYPE *, ' '
    TYPE *, '*** CLOSE ERROR OCCURED IN OPNIF2 ON LOGICAL UNIT',LUN2,
    1 ' ***'
    OPNERR = 1
    WRITE (6,*) '*** EXIT OPNIF2 - CLOSE ERROR TERMINATION ***'
    RETURN
END

```

```

SUBROUTINE MTASEL(STORE,MTAERR,MTAEND)

C
C THIS SUBROUTINE ALLOWS THE USER TO SELECT THE ALGORITHM
C TO BE USED TO MATCH THE INTERESTING POINTS FROM IMAGE ONE
C TO THOSE IN IMAGE TWO.
C
1  TYPE *, '
TYPE *, 'ENTER THE NUMBER CORRESPONDING TO THE ALGORITHM YOU WISH'
TYPE *, 'TO USE TO MATCH THE INTERESTING POINTS:'
TYPE *, '
TYPE *, ' 1. RELAXATION-LABELING'
TYPE *, ' 2. NONE YET'
TYPE *, ' 3. NONE YET'
TYPE *, ' 4. NONE YET'
TYPE *, ' 5. NONE YET'
TYPE *, ' 6. TERMINATE MATCHING ALGORITHM'
TYPE *, ' 7. TERMINATE PROGRAM EXECUTION'
TYPE *, '
TYPE 5
5  FORMAT(5X,'ALGORITHM NUMBER = ',S)
ACCEPT *,MTANBR
IF(MTANBR.LT.1.OR.MTANBR.GT.7)GO TO 100
GO TO(10,20,30,40,50,60,70)MTANBR
10 CALL MTALG1(STORE,MTAERR,MTAEND)
RETURN
20 CALL MTALG2(MTAERR)
GO TO 1
30 CALL MTALG3(MTAERR)
GO TO 1
40 CALL MTALG4(MTAERR)
GO TO 1
50 CALL MTALG5(MTAERR)
GO TO 1
60 RETURN
70 MTAEND = 1
RETURN
100 TYPE *, '
TYPE *, '*** INVALID ALGORITHM NUMBER',MTANBR,' ATTEMPTED ***'
GO TO 1
END

```



```

SUBROUTINE MTALG1(STORE,MTAERR,MTAEND)
C
C THIS SUBROUTINE CONTROLS THE MATCHING PROCESS ASSOCIATED WITH
C THE RELAXATION-LABELING MATCHING ALGORITHM. LABELS ARE DETER-
C MINED FROM THE INITIAL DISPARITIES FORMED BY THE TWO SETS OF
C INTERESTING POINTS. INITIAL PROBABILITIES ARE FORMED BY COMP-
C ARING CORRELATION WINDOWS SURROUNDING EACH INTERESTING POINT
C BASED ON A WEIGHTING ALGORITHM THE USER SELECTS. THESE PRO-
C BABILITIES ARE THEN UPDATED TO FORM THE FINAL PROBABILITIES.
C
C INTEGER CWTEST,W8ANBR
D WRITE (6,*) '*** ENTER MTALG1 ***'
C
C *** DETERMINE THE DISPARITY WINDOW SIZE ***
C
C CALL DWDSIZE
C
C *** DETERMINE THE ORIENTATION OF THE CORRELATION WINDOWS ***
C
C CALL CWCOMP(CWTEST)
C
C *** SELECT THE WEIGHTING ALGORITHM ***
C
C CALL W8ASEL(W8ANBR,MTAEND)
C IF(MTAEND.EQ.1)RETURN
C
C *** DETERMINE THE CONSTANT FOR THE WEIGHING ALGORITHM ***
C
C IF(W8ANBR.EQ.6)GO TO 1
C CALL CONST(C)
C
C *** FORM THE LABELS AND INITIAL WEIGHTS ***
C
C CALL LABELS(STORE,CWTEST,W8ANBR,C,MTAERR,MTAEND)
C IF(MTAERR.EQ.1.OR.MTAEND.EQ.1)RETURN
C
C *** FORM INITIAL PROBABILITIES ***
C
C CALL INPROB(STORE)
C
C *** DETERMINE THE CONSISTENCY NEIGHBORHOOD ***
C
C CALL NGHSIZE
C
C *** DETERMINE THE CONSTANTS AND THRESHOLD FOR UPDATING THE
C LABEL PROBABILITIES ***
C
C CALL CONSTS(A,B)
C CALL THRESH(T)

```

C
C
C
C

*** UPDATE THE LABEL PROBABILITIES ***

CALL UDPROB(STORE,A,B,T)

RETURN
END

```

SUBROUTINE DWDSZE
C
C THIS SUBROUTINE ALLOWS THE USER TO DETERMINE THE NUMBER OF
C ROWS AND COLUMNS OF THE DISPARITY WINDOW TO BE USED TO FORM
C THE LABELS FOR THE MATCHING ALGORITHM. THE USER IS ASKED TO
C CHECK THE INPUT INFORMATION AND MAKE CORRECTIONS IF NECESSARY.
C
COMMON /DSPWND/ DWDROW,DWDCOL,DWDRWH,DWDCLH,DWSZE,DWDINV
INTEGER DWDROW,DWDCOL,DWDRWH,DWDCLH,DWSZE
TYPE *,' '
TYPE *,'PLEASE TYPE THE NUMBER OF ROWS AND COLUMNS'
TYPE *,'OF THE DISPARITY WINDOW:'
TYPE *,' '
C
C *** ENTER THE NUMBER OF ROWS AND COLUMNS OF THE WINDOW ***
C
CALL DRCFND
C
C ***COMPUTE THE WINDOW SIZE (NUMBER OF ROWS X NUMBER OF COLUMNS)***
C
DWSZE = DWDROW*DWDCOL
DWDRWH = DWDROW/2
DWDCLH = DWDCOL/2
C
C *** COMPUTE THE INVERSE OF THE WINDOW SIZE ***
C
DWDINV = 1.0/DWSZE
C
WRITE (6,*) ' '
WRITE (6,*) '*** DISPARITY WINDOW INFORMATION ***'
WRITE (6,*) ' '
WRITE (6,*) 'NUMBER OF DISPARITY WINDOW ROWS = ',DWDROW
WRITE (6,*) 'NUMBER OF DISPARITY WINDOW COLUMNS = ',DWDCOL
WRITE (6,*) 'DISPARITY WINDOW SIZE (ROWSXCOLUMNS) = ',DWSZE
WRITE (6,*) 'INVERSE OF DISPARITY WINDOW SIZE = ',DWDINV
WRITE (6,*) ' '
RETURN
END

```

```

SUBROUTINE DRCFND
C
C THIS SUBROUTINE ALLOWS THE USER TO ENTER THE NUMBR OF ROWS AND
C COLUMNS INTERACTIVELY FOR THE DISPARITY WINDOW TO BE USED TO FIND
C THE LABELS FOR THE MATCHING ALGORITHM. THE USER MAY CORRECT THIS
C INFORMATION IF AN ENTRY ERROR OCCURS.
C
C *** ENTER THE NUMBER OF ROWS AND COLUMNS ***
C
CALL DWROW
CALL DWCOL
C
C *** OPTIONAL ENTRY INFORMATION CHECK ***
C
TYPE *, ' '
TYPE *, 'DO YOU WISH TO CHECK FOR ENTRY ERRORS?'
TYPE *, 'IF YOU DO, TYPE YES. OTHERWISE, TYPE NO.'
TYPE *, ' '
TYPE 1
1 FORMAT(5X, 'ANSWER = ', S)
ACCEPT 2, ANSWER
2 FORMAT(A3)
C
C *** CHECK ANSWER ENTRY ***
C
CALL ANSCHK(ANSWER)
C
IF(ANSWER.EQ.'NO')GO TO 3
C
C *** CHECK ENTRY INFORMATION ***
C
CALL DRCCHK
C
3 RETURN
END

```

```

SUBROUTINE DRCCCHK
C
C THIS SUBROUTINE ALLOWS THE USER TO CHECK THE INFORMATION
C INPUT FOR THE NUMBER OF ROWS AND COLUMNS OF THE DISPARITY
C WINDOW, AND TO CORRECT THEM IF DESIRED.
C
COMMON /DSPWND/ DWDROW,DWDCOL
INTEGER DWDROW,DWDCOL
1 TYPE *, ' '
TYPE *, 'HERE IS WHAT YOU HAVE INPUT:'
TYPE *, ' '
C
C *** DISPLAY THE NUMBER OF WINDOW ROWS AND COLUMNS ***
C
TYPE *, 'NUMBER OF DISPARITY WINDOW ROWS = ', DWDROW
TYPE *, 'NUMBER OF DISPARITY WINDOW COLUMNS = ', DWDCOL
C
C *** VERIFY THE CORRECT RESPONSE ***
C
CALL VERIFY(ANSWER)
C
IF(ANSWER.EQ.'YES')GO TO 2
C
C *** CHANGE INCORRECT INFORMATION ***
C
CALL DRCCOR
C
GO TO 1
2 RETURN
END

```

```

SUBROUTINE DRCCOR
C
C THIS SUBROUTINE ALLOWS THE USER TO CORRECT THE INFORMATION
C INPUT FOR THE NUMBER OF DISPARITY WINDOW ROWS AND COLUMNS.
C
REAL ITEM
C
C *** MAKE ERROR CORRECTIONS ***
C
TYPE *, ' '
TYPE *, 'DO YOU WANT TO CHANGE THE NUMBER OF ROWS, COLUMNS, '
TYPE *, 'OR BOTH?'
1 TYPE *, 'IF YOU WANT TO CHANGE THE NUMBER OF ROWS, TYPE ROWS.'
TYPE *, 'IF YOU WANT TO CHANGE THE NUMBER OF COLUMNS, TYPE COLS.'
TYPE *, 'IF YOU WANT TO CHANGE BOTH, TYPE BOTH.'
TYPE *, ' '
TYPE 2
2 FORMAT(5X, 'ITEM YOU WISH TO CHANGE = ', $)
ACCEPT 3, ITEM
3 FORMAT(A4)
IF(ITEM.EQ.'BOTH')GO TO 10
IF(ITEM.EQ.'ROWS')GO TO 20
IF(ITEM.EQ.'COLS')GO TO 30
TYPE *, ' '
TYPE *, '*** ITEM ENTRY ERROR - PLEASE RETYPE ***'
TYPE *, ' '
GO TO 1
C
C *** CHANGE BOTH NUMBER OF ROWS AND COLUMNS ***
C
10 CALL DWROW
CALL DWCOL
RETURN
C
C *** CHANGE NUMBER OF ROWS ***
C
20 CALL DWROW
RETURN
C
C *** CHANGE NUMBER OF COLUMNS ***
C
30 CALL DWCOL
RETURN
END

```

```

SUBROUTINE DWROW
C
C THIS SUBROUTINE ALLOWS THE USER TO INPUT THE NUMBER OF ROWS
C OF THE DISPARITY WINDOW.
C
C INCLUDE 'COMMON.FTN'
COMMON /DSPWND/ DWDROW
INTEGER DWDROW
C
C *** ENTER THE NUMBER OF WINDOW ROWS ***
C
C TYPE 1
1 FORMAT(5X,'NUMBER OF DISPARITY WINDOW ROWS = ',S)
ACCEPT *,DWDROW
IF(DWDROW.GT.M)GO TO 2
IF(DWDROW.LT.2)GO TO 3
C
C RETURN
2 TYPE *,' '
TYPE *,'*** MAXIMUM WINDOW ROW SIZE',M,' EXCEEDED IN DWROW ***'
GO TO 4
3 TYPE *,' '
TYPE *,'*** ROW SIZE SMALLER THAN 2 (MINIMUM) ATTEMPTED ***'
C
C *** CORRECT ROW SIZE ***
C
4 CALL DRCCOR
C
IF(DWDROW.GT.M)GO TO 2
IF(DWDROW.LT.2)GO TO 3
RETURN
END

```

```

SUBROUTINE DWCOL
C
C THIS SUBROUTINE ALLOWS THE USER TO INPUT THE NUMBER OF COLUMNS
C OF THE DISPARITY WINDOW.
C
INCLUDE 'COMMON.FTN'
COMMON /DSPWND/ X,DWDCOL
INTEGER DWDCOL,X
C
C *** ENTER THE NUMBER OF WINDOW COLUMNS ***
C
TYPE 1
1 FORMAT(5X,'NUMBER OF DISPARITY WINDOW COLUMNS = ',S)
ACCEPT *,DWDCOL
IF(DWDCOL.GT.M)GO TO 2
IF(DWDCOL.LT.2)GO TO 3
C
RETURN
2 TYPE *,' '
TYPE *,'*** MAXIMUM WINDOW COLUMN SIZE',M,' EXCEEDED IN DWCOL ***'
GO TO 4
3 TYPE *,' '
TYPE *,'*** COLUMN SIZE LESS THAN 2 (MINIMUM) ATTEMPTED ***'
C
C *** CORRECT COLUMN SIZE ***
C
CALL DRCCHK
C
IF(DWDCOL.GT.M)GO TO 2
IF(DWDCOL.LT.2)GO TO 3
RETURN
END

```



```

SUBROUTINE CWCOMP(CWTEST)

C
C THIS SUBROUTINE DETERMINES THE ORIENTATION OF ONE CORRELATION
C WINDOW WITH RESPECT TO THE OTHER. IT THEN RETURNS A VALUE
C CORRESPONDING TO THIS ORIENTATION.
C

COMMON /IPS1/ IX1,IY1,CWROW1,CWCOL1
COMMON /IPS2/ IX2,IY2,CWROW2,CWCOL2
INTEGER CWROW1,CWCOL1,CWROW2,CWCOL2,CWTEST
D WRITE (6,*) '*** ENTER CWCOMP ***'
C
C *** DETERMINE RELATIVE CORRELATION WINDOW ORIENTATIONS ***

IF(CWROW1.EQ.CWROW2.AND.CWCOL1.EQ.CWCOL2) CWTEST = 0
IF(CWROW1.GT.CWROW2.AND.CWCOL1.GT.CWCOL2) CWTEST = 1
IF(CWROW1.LT.CWROW2.AND.CWCOL1.LT.CWCOL2) CWTEST = 2
IF(CWROW1.GT.CWROW2.AND.CWCOL1.LT.CWCOL2) CWTEST = 3
IF(CWROW1.LT.CWROW2.AND.CWCOL1.GT.CWCOL2) CWTEST = 4
IF(CWROW1.EQ.CWROW2.AND.CWCOL1.GT.CWCOL2) CWTEST = 5
IF(CWROW1.EQ.CWROW2.AND.CWCOL1.LT.CWCOL2) CWTEST = 6
IF(CWROW1.GT.CWROW2.AND.CWCOL1.EQ.CWCOL2) CWTEST = 7
IF(CWROW1.LT.CWROW2.AND.CWCOL1.EQ.CWCOL2) CWTEST = 8
C

RETURN
END

```

```

SUBROUTINE W8ASEL(W8ANBR,W8AEND)

C
C THIS SUBROUTINE ALLOWS THE USER TO SELECT THE WEIGHTING
C ALGORITHM TO BE USED TO CALCULATE THE WEIGHTS ASSOCIATED
C WITH THE INITIAL PROBABILITIES FOR MATCHING.
C

INTEGER W8ANBR,W8AEND
1  TYPE *,' '
TYPE *,'ENTER THE NUMBER CORRESPONDING TO THE WEIGHTING'
TYPE *,'ALGORITHM YOU WISH TO USE:'
TYPE *,'
TYPE *,' 1. SUM OF SQUARES OF PIXEL DIFFERENCES'
TYPE *,' 2. AVERAGE DIFFERENCE'
TYPE *,' 3. CENTER-MINUS-AVERAGE DIFFERENCE'
TYPE *,' 4. GRADIENT DIFFERENCE'
TYPE *,' 5. VARIANCE DIFFERENCE'
TYPE *,' 6. STATISTICAL CORRELATION'
TYPE *,' 7. TERMINATE PROGRAM EXECUTION'
TYPE *,' '
TYPE 5
5  FORMAT(5X,'WEIGHTING ALGORITHM NUMBER = ',5)
ACCEPT *,W8ANBR
IF(W8ANBR.LT.1.OR.W8ANBR.GT.7)GO TO 10
IF(W8ANBR.EQ.7)W8AEND = 1
RETURN
10 TYPE *,' '
TYPE *,'*** INVALID ALGORITHM NUMBER',W8ANBR,' ATTEMPTED ***'
GO TO 1
END

```

```

SUBROUTINE CONST(C)
C
C THIS SUBROUTINE ALLOWS THE USER TO INPUT AN ARBITRARY
C CONSTANT TO BE USED IN THE WEIGHTING ALGORITHMS.
C
1 TYPE *, '
TYPE *, 'WHAT VALUE OF CONSTANT (C) DO YOU WISH TO USE'
TYPE *, 'FOR THE WEIGHTING ALGORITHMS?'
TYPE *, '
TYPE 2
2 FORMAT(5X, 'WEIGHTING CONSTANT = ', $)
ACCEPT *, C
TYPE *, '
TYPE *, 'HERE IS WHAT YOU HAVE INPUT:'
TYPE *, '
C
C *** DISPLAY THE VALUE OF THE CONSTANT ***
C
TYPE *, 'WEIGHTING CONSTANT = ', C
C
C *** VERIFY THE CORRECT RESPONSE ***
C
CALL VERIFY(ANSWER)
C
IF(ANSWER.EQ.'NO')GO TO 1
WRITE (6,*) '
WRITE (6,*) '*** CONSTANT VALUE FOR WEIGHTING ALGORITHMS ***'
WRITE (6,*) '
WRITE (6,*) 'WEIGHTING CONSTANT = ', C
WRITE (6,*) '
RETURN
END

```

```

SUBROUTINE LABELS(STORE,CWTEST,W8ANBR,C,LABERR,LABEND)

C
C THIS SUBROUTINE COMPUTES THE LABELS AND WEIGHTS FOR THE INITIAL
C PROBABILITIES FOR MATCHING. THE INTERESTING POINTS FROM IMAGE
C TWO WITHIN THE DISPARITY WINDOW OF EACH INTERESTING POINT FROM
C IMAGE ONE ARE ASSIGNED A LABEL OF THE DIFFERENCE BETWEEN THEIR
C ROW AND COLUMN COORDINATE VALUES. A WEIGHT IS THEN ASSIGNED TO
C EACH CANDIDATE MATCH POINT BASED ON THE COMPARISON OF ITS COR-
C RELATION WINDOW WITH THAT OF THE INTERESTING POINT IN IMAGE ONE.
C

INCLUDE 'COMMON.FTN'
VIRTUAL STORE(N,M)
COMMON /IFILE1/ LUN1
COMMON /IFILE2/ LUN2
COMMON /REG1/ IPCW1(M/3),IPCW2(M/3)
COMMON /REG2/ ITMP1(M/3),ITMP2(M/3)
COMMON /IPS1/ IPKNT1,IX,IY,IZ,CWSZE1
COMMON /DSPWND/ DWDROW,DWDCOL,DWDRWH,DWDCLH
INTEGER CWTEST,W8ANBR,W8KNT,CWSZE,CWSZE1,CWSZE2
INTEGER DWDROW,DWDCOL,DWDRWH,DWDCLH,DRWFLG,DCLFLG
INTEGER DRWMIN,DRWMAX,DCLMIN,DCLMAX
D WRITE (6,*) '*** ENTER LABELS ***'
  LBLTST = (2*(N-4))/3 + 3
  A = M/3
  B = SORT(A)
  IROW = B
  MDRTST = DWDROW + 1
  MDCTST = DWDCOL + 1

C
C *** INPUT INTERESTING POINTS FROM IMAGE ONE ***
C
DO 20 J=1,IPKNT1
  READ(LUN1,1,ERR=100) IPROW1,IPCOL1,X1,(IPCW1(I1),I1=1,CWSZE1)
1  FORMAT(2I3,F10.3,<CWSZE1>I3)
  READ(LUN2,2,ERR=200) IPKNT2,IA,IB,IC,CWSZE2
2  FORMAT(5I3)
  STORE(1,J) = IPROW1
  STORE(2,J) = IPCOL1
  W8KNT = LBLTST
  LBLKNT = 0
  I = 2

C
C *** FORM DISPARITY WINDOW ***
C
DRWMIN = IPROW1 - DWDRWH
IF(DRWMIN.LE.0)DRWMIN = 1
DRWMAX = IPROW1 + DWDRWH - MOD(MDRTST,2)
DCLMIN = IPCOL1 - DWDCLH
IF(DCLMIN.LE.0)DCLMIN = 1

```

```

C      DCLMAX = IPCOL1 + DWDCLH - MOD(MDCTST,2)
C
C      *** INPUT INTERESTING POINTS FROM IMAGE TWO ***
C
5     DO 10 K=1,IPKNT2
        DRWFLG = 0
        DCLFLG = 0
        READ(LUN2,3,ERR=200) IPROW2,IPCOL2,X2,(IPCW2(I2),I2=1,CWSZF2)
3     FORMAT(2I3,F10.3,<CWSZF2>I3)
C
C      *** TEST FOR POINTS WITHIN DISPARITY WINDOW ***
C
        IF(IPROW2.GE.DRWMIN.AND.IPROW2.LE.DRWMAX) DRWFLG = 1
        IF(IPCOL2.GE.DCLMIN.AND.IPCOL2.LE.DCLMAX) DCLFLG = 1
        IF(DRWFLG.EQ.0.OR.DCLFLG.EQ.0)GO TO 10
        I = I + 2
        LBLKNT = LBLKNT + 1
        W8KNT = W8KNT + 1
C
C      *** DETERMINE WEIGHT VALUE FOR EACH CANDIDATE MATCH POINT ***
C
        CALL CWDEQ8(CWTEST,CWSZE)
        CALL W8AFND(W8ANBR,CWSZE,C,W,ITMP1,ITMP2,IROW)
        IF(CWTEST.EQ.0)GO TO 4
        BACKSPACE LUN1
        READ(LUN1,1,ERR=100) IPROW1,IPCOL1,X1,(IPCW1(I1),I1=1,CWSZE1)
C
C      *** ASSIGN LABELS TO EACH CANDIDATE POINT ***
C
4     LX = IPROW1 - IPROW2
        LY = IPCOL1 - IPCOL2
D     WRITE (6,*) ' '
D     WRITE (6,*) 'IPROW1 = ',IPROW1,' IPROW2 = ',IPROW2,' LX = ',LX
D     WRITE (6,*) 'IPCOL1 = ',IPCOL1,' IPCOL2 = ',IPCOL2,' LY = ',LY
C
C      *** STORE LABELS ***
C
        STORE(I,J) = LX
        STORE(I+1,J) = LY
C
C      *** STORE WEIGHTS ***
C
        STORE(W8KNT,J) = W
C
C      *** TEST FOR EXCESS OF STORAGE LIMIT ***
C
        IF(I+1.GE.LBLTST)GO TO 30
C
10    CONTINUE
C

```

```

C      *** TEST FOR NO LABELS ***
C
14     IF(LBLKNT.NE.0)GO TO 15
        STORE(W8KNT+1,J) = 0
        LBLKNT = 1
C
C      *** STORE NUMBER OF LABELS ***
C
15     STORE(3,J) = LRLKNT
C
16     REWIND LUN2
20     CONTINUE
        RETURN
30     TYPE *, ' '
        TYPE *, '*** LABEL STORAGE SIZE EXCEEDED ***'
        TYPE *, ' '
        WRITE (6,*) ' '
        WRITE (6,*) '*** LABEL STORAGE SIZE EXCEEDED ***'
        WRITE (6,*) ' '
        WRITE (6,*) 'LABEL STORAGE EXCEEDED AT:'
        WRITE (6,*) ' '
        WRITE (6,*) ' IP NUMBER (SET 1) = ',J
        WRITE (6,*) ' @IP ROW = ',IPROW1
        WRITE (6,*) ' @IP COL = ',IPCOL1
        WRITE (6,*) ' '
        WRITE (6,*) ' IP NUMBER (SET 2) = ',K
        WRITE (6,*) ' @IP ROW = ',IPROW2
        WRITE (6,*) ' @IP COL = ',IPCOL2
        WRITE (6,*) ' '
        DRWMIN = DRWMIN + 1
        DRWMAX = DRWMAX - 1
        DCLMIN = DCLMIN + 1
        DCLMAX = DCLMAX - 1
        DRWNEW = 2*(DRWMAX + MOD(MDRTST,2) - IPROW1 + 1)
        DCLNEW = 2*(DCLMAX + MOD(MDCTST,2) - IPCOL1 + 1)
        IF((DRWMIN+1).GE.DRWMAX.OR.(DCLMIN+1).GE.DCLMAX)GO TO 14
        WRITE (6,*) '*** RETRYING WITH NEW DISPARITY WINDOW ***'
        WRITE (6,*) ' '
        WRITE (6,*) ' NUMBER OF DISPARITY WINDOW ROWS = ',DRWNEW
        WRITE (6,*) ' NUMBER OF DISPARITY WINDOW COLUMNS = ',DCLNEW
        WRITE (6,*) ' '
        REWIND LUN2
        READ(LUN2,2,ERR=200) IPKNT2,IA,IB,IC,CWSZE2
        W8KNT = LBLTST
        LBLKNT = 0
        I = 2
        GO TO 5
100    TYPE *, ' '
        TYPE *, '*** READ ERROR OCCURED IN LABELS ON LOGICAL UNIT',LUN1,
        1' ***'

```

```

CLOSE(UNIT=LUN1,DISPOSE='KEEP')
CLOSE(UNIT=LUN2,DISPOSE='KEEP')
LABERR = 1
WRITE (6,*) '*** EXIT LABELS - READ ERROR TERMINATION ***'
RETURN
200 TYPE *, ' '
TYPE *, '*** READ ERROR OCCURED IN LABELS ON LOGICAL UNIT',LUN2,
1' ***'
CLOSE(UNIT=LUN1,DISPOSE='KEEP')
CLOSE(UNIT=LUN2,DISPOSE='KEEP')
LABERR = 1
WRITE (6,*) '*** EXIT LABELS - READ ERROR TERMINATION ***'
RETURN
END

```

```

SUBROUTINE CWDEQ8(CWTEST,K)

C
C   THIS SUBROUTINE FINDS THE INTERSECTION OF THE CORRELATION
C   WINDOWS BASED ON THE ORIENTATION DETERMINED IN SUBROUTINE
C   CWCOMP. THE COMMON ELEMENTS OF EACH WINDOW ARE THEN RESTORED
C   IN THEIR RESPECTIVE ARRAYS BEGINNING IN COLUMN ONE. THE
C   VALUE OF K RETURNED IS THE COMMON CORRELATION WINDOW SIZE.
C

      INCLUDE 'COMMON.FTN'
      COMMON /REG1/ IPCW1(M/3),IPCW2(M/3)
      COMMON /REG2/ ITMP1(M/3),ITMP2(M/3)
      COMMON /IPS1/ IX1,IY1,CWROW1,CWCOL1,CWSZE1
      COMMON /IPS2/ IX2,IY2,CWROW2,CWCOL2,CWSZE2
      INTEGER CWROW1,CWCOL1,CWSZE1,CWROW2,CWCOL2,CWSZE2,CWTEST
D     WRITE (6,*) '*** ENTER CWDEQ8 ***'
      K = 0
      IF(CWTEST.NE.0)GO TO 5

C
C   *** CASE 0: CWROW1 = CWROW2 AND CWCOL1 = CWCOL2 ***
C
      K = CWSZE1
D     ISZE = CWCOL1
D     GO TO 100
      RETURN

C
5     GO TO(10,20,30,40,50,60,70,80)CWTEST
C
C   *** CASE 1: CWROW1 > CWROW2 AND CWCOL1 > CWCOL2 ***
C
10    IRWDIF = CWROW1 - CWROW2
      IRWDFH = IRWDIF/2
      ICLDIF = CWCOL1 - CWCOL2
      ICLDFH = ICLDIF/2
      DO 15 I=IRWDFH+1,CWROW1-IRWDFH-MOD(IRWDIF,?)
      DO 15 J=ICLDFH+1,CWCOL1-ICLDFH-MOD(ICLDIF,2)
      K = K + 1
      ITMP1(K) = IPCW1((I-1)*CWCOL1+J)
15    CONTINUE
      DO 16 I=1,K
      IPCW1(I) = ITMP1(I)
16    CONTINUE
D     ISZE = CWCOL2
D     GO TO 100
      RETURN

C
C   *** CASE 2: CWROW1 < CWROW2 AND CWCOL1 < CWCOL2 ***
C
20    IRWDIF = CWROW2 - CWROW1
      IRWDFH = IRWDIF/2

```



```

ICLDIF = CWCOL2 - CWCOL1
ICLDFH = ICLDIF/2
DO 25 I=IRWDFH+1,CWROW2-IRWDFH-MOD(IRWDIF,2)
DO 25 J=ICLDFH+1,CWCOL2-ICLDFH-MOD(ICLDIF,2)
K = K + 1
ITMP2(K) = IPCW2((I-1)*CWCOL2+J)
25 CONTINUE
DO 26 I=1,K
IPCW2(I) = ITMP2(I)
26 CONTINUE
D ISZE = CWCOL1
D GO TO 100
RETURN
C
C *** CASE 3: CWROW1 > CWROW2 AND CWCOL1 < CWCOL2 ***
C
30 IRWDIF = CWROW1 - CWROW2
IRWDFH = IRWDIF/2
ICLDIF = CWCOL2 - CWCOL1
ICLDFH = ICLDIF/2
DO 35 I=1,CWROW2
DO 35 J=ICLDFH+1,CWCOL2-ICLDFH-MOD(ICLDIF,2)
K = K + 1
ITMP1(K) = IPCW1(IRWDFH*CWCOL1+K)
ITMP2(K) = IPCW2((I-1)*CWCOL2+J)
35 CONTINUE
DO 36 I=1,K
IPCW1(I) = ITMP1(I)
IPCW2(I) = ITMP2(I)
36 CONTINUE
D ISZE = CWCOL1
D GO TO 100
RETURN
C
C *** CASE 4: CWROW1 < CWROW2 AND CWCOL1 > CWCOL2 ***
C
40 IRWDIF = CWROW2 - CWROW1
IRWDFH = IRWDIF/2
ICLDIF = CWCOL1 - CWCOL2
ICLDFH = ICLDIF/2
DO 45 I=1,CWROW1
DO 45 J=ICLDFH+1,CWCOL1-ICLDFH-MOD(ICLDIF,2)
K = K + 1
ITMP1(K) = IPCW1((I-1)*CWCOL1+J)
ITMP2(K) = IPCW2(IRWDFH*CWCOL2+K)
45 CONTINUE
DO 46 I=1,K
IPCW1(I) = ITMP1(I)
IPCW2(I) = ITMP2(I)
46 CONTINUE

```

```

D      ISZE = CWCOL2
D      GO TO 100
D      RETURN

C
C      *** CASE 5: CWROW1 = CWROW2 AND CWCOL1 > CWCOL2 ***
C
50     ICLDIF = CWCOL1 - CWCOL2
        ICLDFH = ICLDIF/2
        DO 55 I=1,CWROW1
        DO 55 J=ICLDFH+1,CWCOL1-ICLDFH-MOD(ICLDIF,2)
        K = K + 1
        ITMP1(K) = IPCW1((I-1)*CWCOL1+J)
55     CONTINUE
        DO 56 I=1,K
        IPCW1(I) = ITMP1(I)
56     CONTINUE
D      ISZE = CWCOL2
D      GO TO 100
D      RETURN

C
C      *** CASE 6: CWROW1 = CWROW2 AND CWCOL1 < CWCOL2 ***
C
60     ICLDIF = CWCOL2 - CWCOL1
        ICLDFH = ICLDIF/2
        DO 65 I=1,CWROW1
        DO 65 J=ICLDFH+1,CWCOL2-ICLDFH-MOD(ICLDIF,2)
        K = K + 1
        ITMP2(K) = IPCW2((I-1)*CWCOL2+J)
65     CONTINUE
        DO 66 I=1,K
        IPCW2(I) = ITMP2(I)
66     CONTINUE
D      ISZE = CWCOL1
D      GO TO 100
D      RETURN

C
C      *** CASE 7: CWROW1 > CWROW2 AND CWCOL1 = CWCOL2 ***
C
70     IRWDIF = CWROW1 - CWROW2
        IRWDFH = IRWDIF/2
        DO 75 I=IRWDFH+1,CWROW1-IRWDFH-MOD(IRWDIF,2)
        DO 75 J=1,CWCOL1
        K = K + 1
        ITMP1(K) = IPCW1(IRWDFH*CWCOL1+K)
75     CONTINUE
        DO 76 I=1,K
        IPCW1(I) = ITMP1(I)
76     CONTINUE
D      ISZE = CWCOL1
D      GO TO 100

```

```

      RETURN
C
C      *** CASE 8: CWROW1 < CWROW2 AND CWCOL1 = CWCOL2 ***
C
80      IRWDIF = CWROW2 - CWROW1
      IRWDFH = IRWDIF/2
      DO 85 I=IRWDFH+1,CWROW2-IRWDFH-MOD(IRWDIF,2)
      DO 85 J=1,CWCOL1
      K = K + 1
      ITMP2(K) = IPCW2(IRWDFH*CWCOL1+K)
85      CONTINUE
      DO 86 I=1,K
      IPCW2(I) = ITMP2(I)
86      CONTINUE
D      ISZE = CWCOL1
D100     WRITE (6,*) ' '
D      WRITE (6,*) 'NEW CORRELATION WINDOW 1:'
D      WRITE (6,*) '-----'
D      WRITE (6,101)(IPCW1(I),I=1,K)
D101     FORMAT(<ISZE>(2X,I3))
D      WRITE (6,*) ' '
D      WRITE (6,*) 'NEW CORRELATION WINDOW 2:'
D      WRITE (6,*) '-----'
D      WRITE (6,101)(IPCW2(I),I=1,K)
D      WRITE (6,*) ' '
      RETURN
      END

```

```

SUBROUTINE W8AFND(W8ANBR,CWSZE,C,W,ITMP1,ITMP2,IROW)
C
C THIS SUBROUTINE PASSES CONTROL TO THE WEIGHT ALGORITHM SELECTED
C BY THE USER IN SUBROUTINE W8ASEL. THE CORRESPONDING COMPUTED
C WEIGHT IS RETURNED.
C
INTEGER W8ANBR,CWSZE
GO TO(10,20,30,40,50,60)W8ANBR
10 CALL W8ALG1(CWSZE,C,W)
RETURN
20 CALL W8ALG2(CWSZE,C,W)
RETURN
30 CALL W8ALG3(CWSZE,C,W)
RETURN
40 CALL W8ALG4(CWSZE,C,W,ITMP1,ITMP2,IROW)
RETURN
50 CALL W8ALG5(CWSZE,C,W)
RETURN
60 CALL W8ALG6(CWSZE,W)
RETURN
END

```

```

SUBROUTINE W8ALG1(CWSZE,C,W)
C
C   THIS SUBROUTINE FORMS THE SUM OF THE SQUARES OF THE DIFFERENCES
C   OF THE PIXELS OF THE CORRELATION WINDOWS TO BE COMPARED.  A
C   WEIGHT FOR THE MATCHING ALGORITHM IS THEN COMPUTED FROM THIS
C   SUM MULTIPLIED BY AN ARBITRARY CONSTANT.
C
C   INCLUDE 'COMMON.FTN'
COMMON /REG1/ IPCW1(M/3),IPCW2(M/3)
INTEGER CWSZE
D   WRITE (6,*) '*** ENTER W8ALG1 ***'
SUM = 0
C
C   *** FORM SUM OF SQUARES OF PIXEL DIFFERENCES ***
C
DO 10 I=1,CWSZE
CWNDIF = IPCW1(I) - IPCW2(I)
DIFSQR = CWNDIF*CWNDIF
SUM = SUM + DIFSQR
10 CONTINUE
C
C   *** COMPUTE WEIGHT FOR MATCHING ALGORITHM ***
C
W = 1.0/(1.0 + C*SUM)
D   WRITE (6,*) 'W8ALG1 WEIGHT = ',W
RETURN
END

```

```

SUBROUTINE W8ALG2(CWSIZE,C,W)
C
C   THIS SUBROUTINE FORMS THE AVERAGE OF THE ELEMENTS OF THE COR-
C   RELATION WINDOWS TO BE COMPARED.  THE ABSOLUTE VALUE OF THE
C   DIFFERENCE OF THESE AVERAGES IS THEN USED TO COMPUTE A WEIGHT
C   FOR THE MATCHING ALGORITHM.
C
C   INCLUDE 'COMMON.FTN'
COMMON /REG1/ IPCW1(M/3),IPCW2(M/3)
INTEGER CWSIZE
D   WRITE (6,*) '*** ENTER W8ALG2 ***'
SUM1 = 0
SUM2 = 0
C
C   *** FORM SUMS OF PIXELS OF EACH WINDOW ***
C
DO 10 I=1,CWSIZE
SUM1 = SUM1 + IPCW1(I)
SUM2 = SUM2 + IPCW2(I)
10 CONTINUE
C
C   *** FORM WINDOW AVERAGES ***
C
CWAVE1 = SUM1/CWSIZE
CWAVE2 = SUM2/CWSIZE
C
C   *** FORM ABSOLUTE VALUE OF AVERAGE DIFFERENCE ***
C
CWNDIF = ABS(CWAVE1 - CWAVE2)
C
C   *** COMPUTE WEIGHT FOR MATCHING ALGORITHM ***
C
W = 1.0/(1.0 + C*CWNDIF)
D   WRITE (6,*) 'W8ALG2 WEIGHT = ',W
RETURN
END

```

```

SUBROUTINE W8ALG3(CWSIZE,C,W)
C
C   THIS SUBROUTINE FORMS THE CENTER-MINUS-AVERAGE OF THE ELEMENTS
C   OF THE CORRELATION WINDOWS TO BE COMPARED.  THE ABSOLUTE VALUE
C   OF THE DIFFERENCE OF THESE AVERAGES IS THEN USED TO COMPUTE A
C   WEIGHT FOR THE MATCHING ALGORITHM.
C
  INCLUDE 'COMMON.FTN'
  COMMON /REG1/ IPCW1(M/3),IPCW2(M/3)
  INTEGER CWSIZE
D   WRITE (6,*) '*** ENTER W8ALG3 ***'
  SUM1 = 0
  SUM2 = 0
  ICNTR = CWSIZE/2 + 1
C
C   *** FORM SUMS OF PIXELS OF EACH WINDOW ***
C
  DO 10 I=1,CWSIZE
    SUM1 = SUM1 + IPCW1(I)
    SUM2 = SUM2 + IPCW2(I)
10  CONTINUE
C
C   *** FORM WINDOW AVERAGES ***
C
  CWAVE1 = SUM1/CWSIZE
  CWAVE2 = SUM2/CWSIZE
C
C   *** FORM CENTER-MINUS-AVERAGES ***
C
  CMAVE1 = IPCW1(ICNTR) - CWAVE1
  CMAVE2 = IPCW2(ICNTR) - CWAVE2
C
C   *** FORM ABSOLUTE VALUE OF AVERAGE DIFFERENCE ***
C
  CWNDIF = ABS(CMAVE1 - CMAVE2)
C
C   *** COMPUTE WEIGHT FOR MATCHING ALGORITHM ***
C
  W = 1.0/(1.0 + C*CWNDIF)
C
D   WRITE (6,*) 'W8ALG3 WEIGHT = ',W
  RETURN
  END

```

```

SUBROUTINE W8ALG4(ISZE,C,W,ITMP1,ITMP2,K)
C
C THIS SUBROUTINE FORMS THE GRADIENT OF THE ELEMENTS OF THE COR-
C RELATION WINDOWS TO BE COMPARED. THE ABSOLUTE VALUE OF THE
C DIFFERENCE OF THESE GRADIENTS IS THEN USED TO COMPUTE A WEIGHT
C FOR THE MATCHING ALGORITHM.
C
INCLUDE 'COMMON.FTN'
COMMON /REG1/ IPCW1(M/3),IPCW2(M/3)
DIMENSION ITMP1(K,K),ITMP2(K,K)
D WRITE (6,*) '*** ENTER W8ALG4 ***'
C
C *** DETERMINE NEW WINDOW SIZE (SQUARE) ***
C
CWSIZE = ISZE
CWSQRT = SQRT(CWSIZE)
ISQRT = CWSQRT
ISQRTH = ISQRT/2
ISQR = ISQRT*ISQRT
IDIF = CWSIZE - ISQR
IF(IDIF.NE.0)GO TO 1
KOUNT = 1
GO TO 5
1 KOUNT = (IDIF-1)/2 + 2
C
C *** FORM NEW WINDOWS ***
C
5 DO 10 I=1,ISQRT
DO 10 J=1,ISQRT
ITMP1(I,J) = IPCW1(KOUNT)
ITMP2(I,J) = IPCW2(KOUNT)
KOUNT = KOUNT + 1
10 CONTINUE
D WRITE (6,*) ' '
D WRITE (6,*) 'NEW CORRELATION WINDOW 1 (W8ALG4):'
D WRITE (6,*) '-----'
D WRITE (6,11) ((ITMP1(I,J),J=1,ISQRT),I=1,ISQRT)
D11 FORMAT(<ISQRT>(2X,I3))
D WRITE (6,*) ' '
D WRITE (6,*) 'NEW CORRELATION WINDOW 2 (W8ALG4):'
D WRITE (6,*) '-----'
D WRITE (6,11) ((ITMP2(I,J),J=1,ISQRT),I=1,ISQRT)
D WRITE (6,*) ' '
C
C *** FORM SUMS OF UPPER LEFT QUARTER OF WINDOWS ***
C
KOUNT = 0
ASUM1 = 0
ASUM2 = 0

```



```

DO 20 I=1,ISQRT+MOD(ISQRT,2)
DO 20 J=1,ISQRT+MOD(ISQRT,2)
ASUM1 = ASUM1 + ITMP1(I,J)
ASUM2 = ASUM2 + ITMP2(I,J)
KOUNT = KOUNT + 1
20 CONTINUE
C
C *** FORM SUMS OF UPPER RIGHT QUARTER OF WINDOWS ***
C
BSUM1 = 0
BSUM2 = 0
DO 30 I=1,ISQRT+MOD(ISQRT,2)
DO 30 J=ISQRT+MOD(ISQRT,2),ISQRT
BSUM1 = BSUM1 + ITMP1(I,J)
BSUM2 = BSUM2 + ITMP2(I,J)
30 CONTINUE
C
C *** FORM SUMS OF LOWER LEFT QUARTER OF WINDOWS ***
C
CSUM1 = 0
CSUM2 = 0
DO 40 I=ISQRT+MOD(ISQRT,2),ISQRT
DO 40 J=1,ISQRT+MOD(ISQRT,2)
CSUM1 = CSUM1 + ITMP1(I,J)
CSUM2 = CSUM2 + ITMP2(I,J)
40 CONTINUE
C
C *** FORM SUMS OF LOWER RIGHT QUARTER OF WINDOWS ***
C
DSUM1 = 0
DSUM2 = 0
DO 50 I=ISQRT+MOD(ISQRT,2),ISQRT
DO 50 J=ISQRT+MOD(ISQRT,2),ISQRT
DSUM1 = DSUM1 + ITMP1(I,J)
DSUM2 = DSUM2 + ITMP2(I,J)
50 CONTINUE
C
C *** FORM QUADRANT AVERAGES ***
C
A1 = ASUM1/KOUNT
B1 = BSUM1/KOUNT
C1 = CSUM1/KOUNT
D1 = DSUM1/KOUNT
C
A2 = ASUM2/KOUNT
B2 = BSUM2/KOUNT
C2 = CSUM2/KOUNT
D2 = DSUM2/KOUNT
C
C *** FORM COORDINATE DIFFERENCES ***

```

```

C      NDF = (ISQRT+1)/2
      ALPHA1 = ((A1+B1) - (C1+D1))/NDF
      BETA1 = ((B1+D1) - (A1+C1))/NDF
C
      ALPHA2 = ((A2+B2) - (C2+D2))/NDF
      BETA2 = ((B2+D2) - (A2+C2))/NDF
C
C      *** FORM GRADIENT MAGNITUDES ***
C
      G1 = SQRT(ALPHA1*ALPHA1 + BETA1*BETA1)
C
      G2 = SQRT(ALPHA2*ALPHA2 + BETA2*BETA2)
C
C      *** FORM GRADIENT DIFFERENCE ***
C
      CWNDF = ABS(G1 - G2)
C
C      *** FORM WEIGHT FOR MATCHING ALGORITHM ***
C
      W = 1.0/(1.0 + C*CWNDF)
C
D      WRITE (6,*) 'W8ALG4 WEIGHT = ',W
      RETURN
      END

```

```

SUBROUTINE W8ALG5(CWSZE,C,W)
C
C THIS SUBROUTINE FORMS THE VARIANCE OF THE ELEMENTS OF THE COR-
C RELATION WINDOWS TO BE COMPARED. THE ABSOLUTE VALUE OF THE
C DIFFERENCE OF THESE VARIANCES IS THEN USED TO COMPUTE A WEIGHT
C FOR THE MATCHING ALGORITHM.
C
INCLUDE 'COMMON.FTN'
COMMON /REG1/ IPCW1(M/3),IPCW2(M/3)
INTEGER CWSZE
D WRITE (6,*) '*** ENTER W8ALG5 ***'
SUM1 = 0
SUM2 = 0
SQRSM1 = 0
SQRSM2 = 0
C
C *** FORM SQUARES AND SUMS OF PIXELS OF EACH WINDOW ***
C
DO 10 I=1,CWSZE
FLTIC1 = FLOAT(IPCW1(I))
SQRIC1 = FLTIC1*FLTIC1
SQRSM1 = SQRSM1 + SQRIC1
FLTIC2 = FLOAT(IPCW2(I))
SQRIC2 = FLTIC2*FLTIC2
SQRSM2 = SQRSM2 + SQRIC2
SUM1 = SUM1 + IPCW1(I)
SUM2 = SUM2 + IPCW2(I)
10 CONTINUE
C
C *** FORM WINDOW MEANS ***
C
CMEAN1 = SUM1/CWSZE
CMEAN2 = SUM2/CWSZE
C
C *** FORM MEAN SQUARES ***
C
SQRCM1 = CMEAN1*CMEAN1
SQRCM2 = CMEAN2*CMEAN2
C
C *** FORM VARIANCES ***
C
CWVAR1 = SQRSM1/CWSZE - SQRCM1
CWVAR2 = SQRSM2/CWSZE - SQRCM2
C
C *** FORM ABSOLUTE VALUE OF VARIANCE DIFFERENCE ***
C
CVRDIF = ABS(CWVAR1 - CWVAR2)
C
C *** COMPUTE WEIGHT FOR MATCHING ALGORITHM ***

```

C
W = 1.0/(1.0 + C*CVRDIF)
C
D
WRITE (6,*) 'W8ALG5 WEIGHT = ',W
RETURN
END

```

SUBROUTINE W8ALG6(CWSIZE,W)
C
C THIS SUBROUTINE FORMS THE STATISTICAL CORRELATION OF THE ELEMENTS
C OF THE CORRELATION WINDOWS TO BE COMPARED. THE ABSOLUTE VALUE OF
C THE RESULT IS A NUMBER BETWEEN 0 AND 1. THIS NUMBER IS USED AS THE
C WEIGHT FOR THE MATCHING ALGORITHM.
C
INCLUDE 'COMMON.FTN'
COMMON /REG1/ IPCW1(M/3),IPCW2(M/3)
INTEGER CWSIZE
D WRITE (6,*) '*** ENTER W8ALG6 ***'
SUM1 = 0
SUM2 = 0
SORSM1 = 0
SORSM2 = 0
CPSUM = 0
C
C *** FORM SQUARES OF PIXELS OF EACH WINDOW ***
C
DO 10 I=1,CWSIZE
FLTIC1 = FLOAT(IPCW1(I))
SORIC1 = FLTIC1*FLTIC1
SORSM1 = SORSM1 + SORIC1
C
FLTIC2 = FLOAT(IPCW2(I))
SORIC2 = FLTIC2*FLTIC2
SORSM2 = SORSM2 + SORIC2
C
C *** FORM SUMS OF ELEMENTS OF EACH WINDOW ***
C
SUM1 = SUM1 + IPCW1(I)
SUM2 = SUM2 + IPCW2(I)
C
C *** FORM CROSS PRODUCT SUM FOR BOTH WINDOWS ***
C
CRSPRD = FLTIC1*FLTIC2
CPSUM = CPSUM + CRSPRD
C
10 CONTINUE
C
C *** FORM SUM SQUARES ***
C
SORCS1 = SUM1*SUM1
SORCS2 = SUM2*SUM2
C
C *** FORM COVARIANCE ***
C
C12 = CWSIZE*CPSUM - SUM1*SUM2
C

```

```

C      *** FORM INDIVIDUAL VARIANCES ***
C
V1 = SQRS11*CWSZE - SQRC11
V2 = SQRS12*CWSZE - SQRC12
C
C      *** COMPUTE WEIGHT FOR MATCHING ALGORITHM ***
C
W = ABS(C12/SQRT(V1*V2))
C
D      WRITE (6,*) 'W8ALG6 WEIGHT = ',W
      RETURN
      END

```

```

C      SUBROUTINE INPROB(P)
C
C      THIS SUBROUTINE COMPUTES THE INITIAL PROBABILITIES FOR THE
C      MATCHING ALGORITHM FROM THE WEIGHTS COMPUTED AND STORED IN
C      SUBROUTINE LABELS.
C
C      INCLUDE 'COMMON.FTN'
C      VIRTUAL P(N,M)
C      COMMON /IPS1/ IPKNT1
D      WRITE (6,*) '*** ENTER INPROB ***'
C      MINROW = (2*(N-4))/3 + 4
C      DO 30 J=1,IPKNT1
D      WRITE (6,*) ' '
D      WRITE (6,*) 'INITIAL LABEL PROBABILITIES:'
D      WRITE (6,*) '-----'
C      WSUM = 0
C      WMAX = P(MINROW,J)
C      MAXROW = MINROW + P(3,J) - 1
C
C      *** DETERMINE SUM OF WEIGHTS AND MAXIMUM WEIGHT ***
C
C      DO 10 I=MINROW,MAXROW
C      IF(P(I,J).GT.WMAX)WMAX = P(I,J)
C      WSUM = WSUM + P(I,J)
10     CONTINUE
C
C      *** DETERMINE NO MATCH PROBABILITY ***
C
C      P(N,J) = 1.0 - WMAX
C      IF(WSUM.EQ.0)WSUM = 1
C      DO 20 I=MINROW,MAXROW
C
C      *** DETERMINE CONDITIONAL PROBABILITY ***
C
C      CPROB = P(I,J)/WSUM
C
C      *** DETERMINE INITIAL PROBABILITIES ***
C
C      P(I,J) = CPROB*(1.0 - P(N,J))
D      WRITE (6,*) 'P(' ,I ,',',J ,') = ',P(I,J)
20     CONTINUE
D      WRITE (6,*) ' '
D      WRITE (6,*) 'INITIAL NO-MATCH PROBABILITY:'
D      WRITE (6,*) '-----'
D      WRITE (6,*) 'P(' ,N ,',',J ,') = ',P(N,J)
30     CONTINUE
C      RETURN
C      END

```

```

SUBROUTINE NGHSZE
C
C THIS SUBROUTINE ALLOWS THE USER TO DETERMINE THE NUMBER OF
C ROWS AND COLUMNS OF THE NEIGHBORHOOD TO BE USED TO CHECK
C THE LABELS FOR THE MATCHING ALGORITHM. THE USER IS ASKED TO
C CHECK THE INPUT INFORMATION AND MAKE CORRECTIONS IF NECESSARY.
C
COMMON /NBHOOD/ NGHROW,NGHCOL,NGHRWH,NGHCLH,NGSIZE,NGHINV
INTEGER NGHROW,NGHCOL,NGHRWH,NGHCLH,NGSIZE
REAL NGHINV
TYPE *,' '
TYPE *,'PLEASE TYPE THE NUMBER OF ROWS AND COLUMNS'
TYPE *,'OF THE NEIGHBORHOOD:'
TYPE *,' '
C
C *** ENTER THE NUMBER OF ROWS AND COLUMNS OF THE NEIGHBORHOOD ***
C
CALL NRCFND
C
C ***COMPUTE THE NEIGHBORHOOD(NUMBER OF ROWS X NUMBER OF COLUMNS)***
C
NGSIZE = NGHROW*NGHCOL
NGHRWH = NGHROW/2
NGHCLH = NGHCOL/2
C
C *** COMPUTE THE INVERSE OF THE NEIGHBORHOOD ***
C
NGHINV = 1.0/NGSIZE
C
WRITE (6,*) ' '
WRITE (6,*) '*** NEIGHBORHOOD INFORMATION ***'
WRITE (6,*) ' '
WRITE (6,*) 'NUMBER OF NEIGHBORHOOD ROWS = ',NGHROW
WRITE (6,*) 'NUMBER OF NEIGHBORHOOD COLUMNS = ',NGHCOL
WRITE (6,*) 'NEIGHBORHOOD SIZE (ROWSXCOLUMNS) = ',NGSIZE
WRITE (6,*) 'INVERSE OF NEIGHBORHOOD SIZE = ',NGHINV
WRITE (6,*) ' '
RETURN
END

```



```

SUBROUTINE NRCFND
C
C THIS SUBROUTINE ALLOWS THE USER TO ENTER THE NUMBER OF ROWS AND
C COLUMNS INTERACTIVELY FOR THE NEIGHBORHOOD TO BE USED TO CHECK
C THE LABELS FOR THE MATCHING ALGORITHM. THE USER MAY CORRECT THIS
C INFORMATION IF AN ENTRY ERROR OCCURS.
C
C *** ENTER THE NUMBER OF ROWS AND COLUMNS ***
C
CALL NGROW
CALL NGCOL
C
C *** OPTIONAL ENTRY INFORMATION CHECK ***
C
TYPE *, ' '
TYPE *, 'DO YOU WISH TO CHECK FOR ENTRY ERRORS?'
TYPE *, 'IF YOU DO, TYPE YES. OTHERWISE, TYPE NO.'
TYPE *, ' '
TYPE 1
1 FORMAT(5X, 'ANSWER = ', $)
ACCEPT 2, ANSWER
2 FORMAT(A3)
C
C *** CHECK ANSWER ENTRY ***
C
CALL ANSCHK(ANSWER)
C
IF(ANSWER.EQ.'NO')GO TO 3
C
C *** CHECK ENTRY INFORMATION ***
C
CALL NRCCHK
C
3 RETURN
END

```

```

C      SUBROUTINE NRCCHK
C
C      THIS SUBROUTINE ALLOWS THE USER TO CHECK THE INFORMATION
C      INPUT FOR THE NUMBER OF ROWS AND COLUMNS OF THE NEIGHBOR-
C      HOOD, AND TO CORRECT THEM IF DESIRED.
C
C      COMMON /NBHOOD/ NGHROW,NGHCOL
C      INTEGER NGHROW,NGHCOL
1     TYPE *, ' '
C      TYPE *, 'HERE IS WHAT YOU HAVE INPUT:'
C      TYPE *, ' '
C
C      *** DISPLAY THE NUMBER OF NEIGHBORHOOD ROWS AND COLUMNS ***
C
C      TYPE *, 'NUMBER OF NEIGHBORHOOD ROWS = ', NGHROW
C      TYPE *, 'NUMBER OF NEIGHBORHOOD COLUMNS = ', NGHCOL
C
C      *** VERIFY THE CORRECT RESPONSE ***
C
C      CALL VERIFY(ANSWER)
C
C      IF(ANSWER.EQ.'YES')GO TO 2
C
C      *** CHANGE INCORRECT INFORMATION ***
C
C      CALL NRCCOR
C
C      GO TO 1
2     RETURN
C      END

```

```

SUBROUTINE NRCCOR
C
C THIS SUBROUTINE ALLOWS THE USER TO CORRECT THE INFORMATION
C INPUT FOR THE NUMBER OF NEIGHBORHOOD ROWS AND COLUMNS.
C
REAL ITEM
C
C *** MAKE ERROR CORRECTIONS ***
C
TYPE *, ' '
TYPE *, 'DO YOU WANT TO CHANGE THE NUMBER OF ROWS, COLUMNS,'
TYPE *, 'OR BOTH?'
1 TYPE *, 'IF YOU WANT TO CHANGE THE NUMBER OF ROWS, TYPE ROWS.'
TYPE *, 'IF YOU WANT TO CHANGE THE NUMBER OF COLUMNS, TYPE COLS.'
TYPE *, 'IF YOU WANT TO CHANGE BOTH, TYPE BOTH.'
TYPE *, ' '
TYPE 2
2 FORMAT(5X, 'ITEM YOU WISH TO CHANGE = ', S)
ACCEPT 3, ITEM
3 FORMAT(A4)
IF (ITEM.EQ. 'BOTH') GO TO 10
IF (ITEM.EQ. 'ROWS') GO TO 20
IF (ITEM.EQ. 'COLS') GO TO 30
TYPE *, ' '
TYPE *, '*** ITEM ENTRY ERROR - PLEASE RETYPE ***'
TYPE *, ' '
GO TO 1
C
C *** CHANGE BOTH NUMBER OF ROWS AND COLUMNS ***
C
10 CALL NGROW
CALL NGCOL
RETURN
C
C *** CHANGE NUMBER OF ROWS ***
C
20 CALL NGROW
RETURN
C
C *** CHANGE NUMBER OF COLUMNS ***
C
30 CALL NGCOL
RETURN
END

```

```

C      SUBROUTINE NGROW
C
C      THIS SUBROUTINE ALLOWS THE USER TO INPUT THE NUMBER OF ROWS
C      OF THE NEIGHBORHOOD.
C
C      INCLUDE 'COMMON.FTN'
C      COMMON /NBHOOD/ NGHROW
C
C      *** ENTER THE NUMBER OF NEIGHBORHOOD ROWS ***
C
C      TYPE 1
1      FORMAT(5X,'NUMBER OF NEIGHBORHOOD ROWS = ',5)
      ACCEPT *,NGHROW
      IF(NGHROW.GT.1)GO TO 2
      IF(NGHROW.LT.2)GO TO 3
C
      RETURN
2      TYPE *,' '
      TYPE *,'*** MAXIMUM NEIGHBORHOOD ROW SIZE',M,' EXCEEDED IN NGROW **
      1*'
      GO TO 4
3      TYPE *,' '
      TYPE *,'*** ROW SIZE SMALLER THAN 2 (MINIMUM) ATTEMPTED ***'
C
C      *** CORRECT ROW SIZE ***
C
4      CALL NRCCOR
C
      IF(NGHROW.GT.M)GO TO 2
      IF(NGHROW.LT.2)GO TO 3
      RETURN
      END

```

```

C      SUBROUTINE NGCOL
C
C      THIS SUBROUTINE ALLOWS THE USER TO INPUT THE NUMBER OF COLUMNS
C      OF THE NEIGHBORHOOD.
C
C      INCLUDE 'COMMON.FTN'
C      COMMON /NBHOOD/ IX,NGHCOL
C
C      *** ENTER THE NUMBER OF NEIGHBORHOOD COLUMNS ***
C
C      TYPE 1
1      FORMAT(5X,'NUMBER OF NEIGHBORHOOD COLUMNS = ',5)
      ACCEPT *,NGHCOL
      IF(NGHCOL.GT.M)GO TO 2
      IF(NGHCOL.LT.2)GO TO 3
C
C      RETURN
2      TYPE *,' '
      TYPE *,'*** MAXIMUM NEIGHBORHOOD COLUMN SIZE',M,' EXCEEDED IN NGCO
      IL ***'
      GO TO 4
3      TYPE *,' '
      TYPE *,'*** COLUMN SIZE LESS THAN 2 (MINIMUM) ATTEMPTED ***'
C
C      *** CORRECT COLUMN SIZE ***
C
4      CALL WRCCHK
C
      IF(NGHCOL.GT.M)GO TO 2
      IF(NGHCOL.LT.2)GO TO 3
      RETURN
      END

```

```

SUBROUTINE CONSTS(A,B)
C
C THIS SUBROUTINE ALLOWS THE USER TO INPUT TWO ARBITRARY
C CONSTANTS TO BE USED IN THE MATCHING ALGORITHM.
C
1 TYPE *, ' '
TYPE *, 'WHAT VALUE OF CONSTANTS (A,B) DO YOU WISH TO USE'
TYPE *, 'FOR THE MATCHING ALGORITHM?'
TYPE *, ' '
TYPE 2
2 FORMAT(5X, 'A MATCHING CONSTANT = ', S)
ACCEPT *, A
TYPE 3
3 FORMAT(5X, 'B MATCHING CONSTANT = ', S)
ACCEPT *, B
TYPE *, ' '
TYPE *, 'HERE IS WHAT YOU HAVE INPUT:'
TYPE *, ' '
C
C *** DISPLAY THE VALUE OF THE CONSTANTS ***
C
TYPE *, 'A MATCHING CONSTANT = ', A
TYPE *, 'B MATCHING CONSTANT = ', B
C
C *** VERIFY THE CORRECT RESPONSE ***
C
CALL VERIFY(ANSWER)
C
IF(ANSWER.EQ.'NO')GO TO 1
WRITE (6,*) ' '
WRITE (6,*) '*** VALUE OF CONSTANTS FOR MATCHING ALGORITHM ***'
WRITE (6,*) ' '
WRITE (6,*) 'A MATCHING CONSTANT = ', A
WRITE (6,*) 'B MATCHING CONSTANT = ', B
WRITE (6,*) ' '
RETURN
END

```

```

SUBROUTINE THRESH(T)
C
C THIS SUBROUTINE ALLOWS THE USER TO INPUT A THRESHOLD
C VALUE USED TO COMPARE THE DISPARITY LABELS FOR THE
C MATCHING ALGORITHM.
C
1 TYPE *, ' '
TYPE *, 'WHAT VALUE OF THRESHOLD DO YOU WISH TO USE'
TYPE *, 'TO COMPARE THE DISPARITY LABELS?'
TYPE *, ' '
TYPE 2
2 FORMAT(5X, 'DISPARITY LABEL THRESHOLD = ', S)
ACCEPT *, T
TYPE *, ' '
TYPE *, 'HERE IS WHAT YOU HAVE INPUT:'
TYPE *, ' '
C
C *** DISPLAY THE VALUE OF THRESHOLD ***
C
TYPE *, 'DISPARITY LABEL THRESHOLD = ', T
C
C *** VERIFY THE CORRECT RESPONSE ***
C
CALL VERIFY(ANSWER)
C
IF(ANSWER.EQ.'NO')GO TO 1
WRITE (6,*) ' '
WRITE (6,*) '*** THRESHOLD FOR DISPARITY LABELS ***'
WRITE (6,*) ' '
WRITE (6,*) 'DISPARITY LABEL THRESHOLD = ', T
WRITE (6,*) ' '
RETURN
END

```

```

C      SUBROUTINE UDPROB(P,A,B,T)
C
C      THIS SUBROUTINE UPDATES THE INITIAL PROBABILITIES DETERMINED
C      BY SUBROUTINE INPROB.  THESE UPDATED PROBABILITIES ARE BASED
C      ON THE CONSISTENCY OF THE LABELS OF CANDIDATE NODES WITHIN
C      A CERTAIN NEIGHBORHOOD SPECIFIED BY THE USER.  IF THERE ARE
C      MANY NODES WITHIN A CERTAIN NEIGHBORHOOD WITH SIMILAR LABELS,
C      THE PROBABILITY OF A MATCH FOR THAT LABEL INCREASES.
C
C      INCLUDE 'COMMON.FTN'
C      VIRTUAL P(N,M)
C      COMMON /IPS1/ IPKNT1
C      COMMON /REG1/ QL(((2*(N-4))/3)/2)
C      COMMON /NBHOOD/ NGHROW,NGHCOL,NGHRWH,NGHCLH
C      INTEGER QKNT,PKNT,PROW
D      WRITE (6,*) '*** ENTER UDPROB ***'
C      PROW = (2*(N-4))/3 + 3
C      MDRTST = NGHROW + 1
C      MDCTST = NGHCOL + 1
C      DO 50 ITER=1,10
D      WRITE (6,*) ' '
D      WRITE (6,*) '*****'
D      WRITE (6,*) '* *'
D      WRITE (6,*) '* ITERATION NUMBER ',ITER,' *'
D      WRITE (6,*) '* *'
D      WRITE (6,*) '*****'
C      DO 50 J=1,IPKNT1
C
C      *** INITIALIZE Q(L) AND Q(L*) ***
C
C      NLBLS1 = P(3,J)
C      DO 10 JJ=1,NLBLS1
10      QL(JJ) = 0
C      CONTINUE
C      QLSTR = 0
C
C      *** DETERMINE NEIGHBORHOOD FOR LABEL CONSISTENCY ***
C
C      NROW1 = P(1,J)
C      NCOL1 = P(2,J)
C      NRWMIN = NROW1 - NGHRWH
C      IF(NRWMIN.LE.0)NRWMIN = 1
C      NRWMAX = NROW1 + NGHRWH - MOD(MDRTST,2)
C      NCLMIN = NCOL1 - NGHCLH
C      IF(NCLMIN.LE.0)NCLMIN = 1
C      NCLMAX = NCOL1 + NGHCLH - MOD(MDCTST,2)
C
C      MINPRW = PROW + 1
C      MAXPRW = PROW + NLBLS1

```



```

MAXRW1 = 2*NLBLS1 + 3
C
C *** TEST EACH CANDIDATE NODE (K NOT = J) ***
C
DO 20 K=1,IPKNT1
IF(K.EQ.J)GO TO 20
QKNT = 0
PSUM = 0
NRWFLG = 0
NCLFLG = 0
C
C *** TEST FOR CANDIDATE NODES WITHIN CONSISTENCY NEIGHBORHOOD ***
C
NROW2 = P(1,K)
NCOL2 = P(2,K)
IF(NROW2.GE.NRWMIN.AND.NROW2.LE.NRWMAX) NRWFLG = 1
IF(NCOL2.GE.NCLMIN.AND.NCOL2.LE.NCLMAX) NCLFLG = 1
IF(NRWFLG.EQ.0.OR.NCLFLG.EQ.0)GO TO 20
C
C *** UPDATE NO-MATCH CONSISTENCY PROPERTY Q(L*) ***
C
QLSTR = QLSTR + P(N,K)
C
LBRFLG = 0
LBCFLG = 0
NLBLS2 = P(3,K)
MAXRW2 = 2*NLBLS2 + 3
C
C *** TEST LABELS OF CANDIDATE NODES FOR CONSISTENCY ***
C
DO 20 I1=4,MAXRW1,2
PKNT = PROW
QKNT = QKNT + 1
DO 20 I2=4,MAXRW2,2
PKNT = PKNT + 1
C
C *** DETERMINE LABEL DIFFERENCES ***
C
LBLRDF = ABS(P(I1,J) - P(I2,K))
LBLCDF = ABS(P(I1+1,J) - P(I2+1,K))
C
C *** TEST LABEL DIFFERENCES AGAINST THRESHOLD ***
C
IF(LBLRDF.LE.T) LBRFLG = 1
IF(LBLCDF.LE.T) LBCFLG = 1
IF(LBRFLG.EQ.0.OR.LBCFLG.EQ.0)GO TO 15
C
C *** UPDATE MATCH CONSISTENCY PROPERTY Q(L) ***
C
QL(QKNT) = QL(QKNT) + P(PKNT,K)

```

```

C
15  L3RFLG = 0
    L3CFLG = 0
20  CONTINUE
C
C    *** UPDATE LABEL PROBABILITIES ***
C
    IQ = 0
    DO 30 I=MINPRW,MAXPRW
    IQ = IQ + 1
    P(I,J) = P(I,J)*(A + B*QL(IQ))
    PSUM = PSUM + P(I,J)
30  CONTINUE
C
C    *** UPDATE NO-MATCH PROBABILITY ***
C
    P(N,J) = P(N,J)*(A + B*QLSTR)
    PSUM = PSUM + P(N,J)
C
C    *** OBTAIN NEW LABEL PROBABILITIES ***
C
    DO 40 I=MINPRW,MAXPRW
    P(I,J) = P(I,J)/PSUM
40  CONTINUE
C
C    *** OBTAIN NEW NO-MATCH PROBABILITY ***
C
    P(N,J) = P(N,J)/PSUM
C
D    WRITE (6,*) ' '
D    WRITE (6,*) 'NEW LABEL PROBABILITIES:'
D    WRITE (6,*) '-----'
D    DO 45 I=MINPRW,MAXPRW
D    WRITE (6,*) 'P(' ,I,',',J,') = ',P(I,J)
D45  CONTINUE
D    WRITE (6,*) ' '
D    WRITE (6,*) 'NEW NO-MATCH PROBABILITY:'
D    WRITE (6,*) '-----'
D    WRITE (6,*) 'P(' ,N,',',J,') = ',P(N,J)
D    WRITE (6,*) ' '
50  CONTINUE
    RETURN
    END

```

SUBROUTINE MTALG2(MTAERR)
TYPE *, '*** MTALG2 NOT AVAILABLE AT THIS TIME ***'
RETURN
END

SUBROUTINE MTALG3(MTAERR)
TYPE *, '*** MTALG3 NOT AVAILABLE AT THIS TIME ***'
RETURN
END

SUBROUTINE MTALG4(MTAERR)
TYPE *, '*** MTALG4 NOT AVAILABLE AT THIS TIME ***'
RETURN
END

SUBROUTINE MTALG5(MTAERR)
TYPE *, '*** MTALG5 NOT AVAILABLE AT THIS TIME ***'
RETURN
END

```

SUBROUTINE MPLST(P)
C
C THIS SUBROUTINE DETERMINES THE POINTS IN IMAGE 2 WHICH ARE
C MATCHED WITH THOSE IN IMAGE 1. IF THE PROBABILITY OF ANY
C NODE IS GREATER THAN OR EQUAL TO 0.7, THAT POINT IS DEFINED
C TO BE MATCHED. THE POINT IN IMAGE 2 WHICH IS MATCHED TO
C THAT IN IMAGE 1 IS DETERMINED BY ADDING THE CORRESPONDING
C LABEL TO THE POINT IN IMAGE 1 (THE INTERESTING POINT). AFTER
C ALL THE MATCHED POINTS ARE DETERMINED, THEY ARE LISTED ALONG
C WITH SOME MATCH STATISTICS.
C
INCLUDE 'COMMON.FTN'
VIRTUAL P(N,M)
COMMON /IPS1/ IPKNT1
COMMON /IPS2/ IPKNT2
MINROW = (2*(N-4))/3 + 4
MPKNT = 0
WRITE (6,*) ' '
WRITE (6,*) ' '
WRITE (6,1)
1 FORMAT(1X,67('-'))
DO 10 J=1,IPKNT1
NLBLS = P(3,J)
LBLKNT = 2
MAXROW = MINROW + NLBLS - 1
DO 10 I=MINROW,MAXROW
LBLKNT = LBLKNT + 2
C
C *** TEST FOR MATCHED POINTS ***
C
IF(P(I,J).LT.0.7)GO TO 10
C
MPKNT = MPKNT + 1
C
C *** DETERMINE COORDINATES OF MATCHED POINT IN IMAGE 2 ***
C
IPROW = P(1,J)
IPCOL = P(2,J)
LX = P(LBLKNT,J)
LY = P(LBLKNT+1,J)
MPROW = IPROW - LX
MPCOL = IPCOL - LY
C
WRITE (6,5) J,IPROW,IPCOL,MPROW,MPCOL
5 FORMAT(2X,'IP(',I3,'): NODE(',I3,',',I3,') ON IMAGE 1 ----> NODE(
1',I3,',',I3,') ON IMAGE 2')
WRITE (6,6) LX,LY
6 FORMAT(5X,'LX = ',I3,5X,'LY = ',I3)
10 CONTINUE

```

```

WRITE (6,*) ' '
WRITE (6,*) ' *** MATCH STATISTICS ***'
WRITE (6,11)
11 FORMAT(IX,51('-'))
WRITE (6,*) ' NUMBER OF INTERESTING POINTS IN IMAGE 1 = ',IPKNT1
WRITE (6,*) ' NUMBER OF INTERESTING POINTS IN IMAGE 2 = ',IPKNT2
WRITE (6,*) ' NUMBER OF MATCHED POINTS = ',MPKNT
WRITE (6,*) ' NUMBER OF UNMATCHED POINTS = ',IPKNT1-MPKNT
WRITE (6,*) ' '
RATIO = FLOAT(MPKNT)/FLOAT(IPKNT1)
WRITE (6,*) ' PERCENTAGE MATCHED = ',RATIO*100.0
WRITE (6,*) ' '
RETURN
END

```

Appendix F: Image Display Section

This appendix contains the FORTRAN source listing for the Image Display Section of program DIDA.

```

      SUBROUTINE IMGDSP(IMGERR,IMGEND)
C
C      THIS SUBROUTINE ALLOWS THE USER TO DISPLAY AN IMAGE ON A CRT
C      USING THE RAMTEK DRIVER.  THE USER IS ASKED WHICH IMAGE DISPLAY
C      OPERATION IS DESIRED.  THAT OPERATION IS THEN PERFORMED.
C
      COMMON /FILE/ LUN
      COMMON /RBUF/ IPAR(6),IOSTAT(2)
      WRITE (6,*) '*** ENTER IMGDSP ***'
C
      *** DEFINE IMAGE SIZE ***
      CALL IMGSZ
C
      *** ENTER THE LOGICAL UNIT NUMBER AND FILE NAME OF THE IMAGE
      FILE TO BE INPUT ***
C
      TYPE *, ' '
      TYPE *, 'ENTER THE LOGICAL UNIT NUMBER AND FILE NAME FOR THE'
      TYPE *, 'IMAGE YOU WISH TO DISPLAY:'
      CALL FLUFND
C
      *** OPEN THE IMAGE FILE ***
      CALL OPNOLD
C
      DETERMINE THE LOGICAL UNIT NUMBER FOR THE RAMTEK DRIVER ***
      LUNRM = LUN + 1
      IF(LUNRM.EQ.5)LUNRM = 7
      IF(LUNRM.GT.10)LUNRM = 1
C
      *** ASSIGN LUNRM TO THE RAMTEK DRIVER ***
      CALL ASNLUN(LUNRM,'RM:',0)
C
      *** SELECT THE IMAGE DISPLAY OPERATION DESIRED ***
      CALL DSPSEL(LUNRM,IMGERR,IMGEND)
      IF(IMGERR.EQ.1.OR.IMGEND.EQ.1)GO TO 10
C
      RETURN
```

```

C      *** CLOSE THE IMAGE FILE ***
C
C      CALL CLOSKP(IMGERR)
C
C      RETURN
C      END

C      SUBROUTINE RMIOER(IOB,IER)
C
C      TIHS SUBROUTINE WRITES AN ERROR MESSAGE IF AN ERROR OCCURS WHEN
C      USING THE RAMTEK DRIVER.
C
C      BYTE IOB
C      TYPE IO,IER,IOB
C      WRITE (6,10) IER,IOB
10     FORMAT(/,' *** RAMTEK IO ERROR MESSAGE ***',/,
1' # DIRECTIVE STATUS = ',I4,/, ' # IOSTAT = ',I4,/)
C      RETURN
C      END

```

```

SUBROUTINE DSPSEL(LUN,DSPERR,DSPEND)

C
C THIS SUBROUTINE ALLOWS THE USER TO SELECT THE IMAGE DISPLAY
C OPERATION DESIRED. THE IMAGE DISPLAY IS THEN PERFORMED.
C

INTEGER DSPERR,DSPEND,DSPNBR
1  TYPE *, '
TYPE *, 'ENTER THE NUMBER CORRESPONDING TO THE IMAGE DISPLAY'
TYPE *, 'OPERATION YOU WISH TO PERFORM:'
TYPE *, '
TYPE *, ' 1. DISPLAY BLACK AND WHITE IMAGE'
TYPE *, ' 2. DISPLAY COLOR IMAGE'
TYPE *, ' 3. DISPLAY INTERESTING POINTS ON A'
TYPE *, '    BLACK AND WHITE IMAGE'
TYPE *, ' 4. DISPLAY INTERESTING POINTS ON A'
TYPE *, '    COLOR IMAGE'
TYPE *, ' 5. NONE YET'
TYPE *, ' 6. TERMINATE IMAGE DISPLAY OPERATIONS'
TYPE *, ' 7. TERMINATE PROGRAM EXECUTION'
TYPE *, '
TYPE 5
5  FORMAT(5X,'IMAGE DISPLAY NUMBER = ',S)
ACCEPT *,DSPNBR
IF(DSPNBR.LT.1.OR.DSPNBR.GT.7)GO TO 100
GO TO(10,20,30,40,50,60,70)DSPNBR
10  ICOLOR = 0
CALL RAMST(LUN,ICOLOR,DSPERR)
IF(DSPERR.EQ.1)RETURN
CALL IMDSP1(LUN,ISCALE,DSPERR)
RETURN
20  ICOLOR = 1
CALL RAMST(LUN,ICOLOR,DSPERR)
IF(DSPERR.EQ.1)RETURN
CALL IMDSP1(LUN,ISCALE,DSPERR)
RETURN
30  ICOLOR = 0
CALL RAMST(LUN,ICOLOR,DSPERR)
IF(DSPERR.EQ.1)RETURN
CALL IMDSP2(LUN,DSPERR)
RETURN
40  ICOLOR = 1
CALL RAMST(LUN,ICOLOR,DSPERR)
IF(DSPERR.EQ.1)RETURN
CALL IMDSP2(LUN,DSPERR)
RETURN
50  TYPE *, '*** OPTION 5 NOT AVAILABLE AT THIS TIME ***'
GO TO 1
60  RETURN
70  DSPEND = 1

```


100 RETURN
 TYPE *, ' '
 TYPE *, '*** INVALID OPERATION NUMBER', DSPNBR, ' ATTEMPTED ***'
 GO TO 1
 END

```

SUBROUTINE RAMST(LUN,ICOLOR,RAMERR)
C
C THIS SUBROUTINE INITIALIZES THE VIDEO SCREEN USING THE RAMTEK
C DRIVER. IT ALSO INITIALIZES THE VIDEO LOOKUP TABLE (VLT).
C
INCLUDE 'COMMON.FTN'
COMMON /RBUFF/ IPAR(6),IOSTAT(2)
DIMENSION IVLT(256),IMBUF(260)
LOGICAL*1 IBYTE
EQUIVALENCE (IMBUF(5),IVLT(1)),(IOSTAT(1),IByte)
INTEGER RAMERR
WRITE (6,*) '*** ENTER RAMST ***'
D
C
C *** INITIALIZE VIDEO LOOKUP TABLE ***
C
IMBUF(1) = "2400
IMBUF(2) = "1400
IMBUF(3) = 0
IMBUF(4) = 514
C
C *** INITIALIZE REMAINDER OF TABLE ***
C
IF (ICOLOR .EQ. 0) GO TO 20
C
C *** SET VLT TO COLOR VALUES
C
DO 30 I =1,32
IVLT(I) = 0
IVLT(I+32) = "17
IVLT(I+64) = "7417
IVLT(I+96) = "7400
IVLT(I+128) = "7560
IVLT(I+160) = "7760
IVLT(I+192) = "3760
IVLT(I+224) = "360
30 CONTINUE
IVLT(257) = 0
GO TO 40
C
C *** SET TO BLACK AND WHITE
C
20 DO 10 I=1,16
DO 10 J=1,16
IVLT(16*(I-1)+J) = 273*(I-1)
10 CONTINUE
IVLT(257) = 4080
C
C *** INITIALIZE QIO BUFFER ***
C

```

```

40      CALL GETADR(IPAR,IMBUF)
        IPAR(2) = IMBUF(4) + 8
        IPAR(3) = 0

C
C      *** SEND INITIALIZATION PARAMETERS TO RAMTEK DRIVER ***
C
        CALL WTOIO("410,LUN,2,3,IOSTAT,IPAR,RAMERR)
        IF(RAMERR.EQ.1)RAMERR = 0
        IF(RAMERR.GE.0)RETURN

C
C      *** WRITE RAMTEK ERROR MESSAGE ***
C
        CALL RMIOER(1BYTE,RAMERR)
        RAMERR = 1

C
        RETURN
        END

```

```

C      SUBROUTINE IMDSP1(LUN,ISCALE,IMDERR)
C
C      THIS SUBROUTINE DISPLAYS A BLACK AND WHITE IMAGE ON A CRT USING
C      THE RAMTEK DRIVER.
C
C      INCLUDE 'COMMON.FTN'
C      COMMON /BUF/ IOBUFR(L,M)
C      COMMON /INBUF/ IX(8),INTBUF(M)
C      COMMON /IMAGE/ IMGROW,IMGCOL
C      LOGICAL*1 IOBUFR
C      INTEGER RECNUM
D      WRITE (6,*) '*** ENTER IMDSP1 ***'
C      I = 1
C      ISCALE = 1
C      TYPE *, ' '
C      TYPE *, 'DO YOU WISH TO SCALE THE IMAGE UP TO A SIZE OF '
C      TYPE *, '(512X512) FROM ITS PRESENT SIZE?'
C      TYPE *, ' '
C      TYPE *, 'IF YOU DO, TYPE YES. OTHERWISE, TYPE NO.'
C      TYPE *, ' '
C      TYPE 1
1      FORMAT(5X,'ANSWER = ',S)
C      ACCEPT 2,ANSWER
2      FORMAT(A3)
C      CALL ANSCHK(ANSWER)
C      IF(ANSWER.EQ.'YES')ISCALE = 0
C      DO 10 RECNUM=1,IMGROW
C
C      *** INPUT AN IMAGE ROW INTO THE INPUT BUFFER ***
C
C      CALL INPUT(RECNUM,I,IMDERR)
C      IF(IMDERR.EQ.1)RETURN
C
C      *** PERFORM BYTE-TO-WORD NUMBER CONVERSION ***
C
C      DO 5 J=1,IMGCOL
C      INTBUF(J) = IOBUFR(1,J)
C      IF(INTBUF(J).LT.0)INTBUF(J) = INTBUF(J) + 256
5      CONTINUE
C
C      *** SEND IMAGE ROW TO RAMTEK DRIVER ***
C
C      CALL RAMWT(LUN,ISCALE,RECNUM,IMDERR)
C      IF(IMDERR.EQ.1)RETURN
C
C      CONTINUE
10
C
C      *** CLOSE THE IMAGE FILE ***
C

```

C

CALL CLOSKP(INDERR)

RETURN
END

```

SUBROUTINE RAMUT(LUN, ISCALE, RECNUM, RAMERR)
C
C THIS SUBROUTINE WRITES ONE ROW OF AN IMAGE TO THE
C VIDEO SCREEN.
C
  INCLUDE 'COMMON.FTN'
  COMMON /IMAGE/ IMGROW, IMGCOL
  COMMON /RBUF/ IPAR(6), IOSTAT(2)
  COMMON /IBUF/ IX(8), INTRBUF(M)
  DIMENSION IMBUF(M+10)
  LOGICAL*1 IBYTE
  EQUIVALENCE (IMBUF(9), INTRBUF(1)), (IOSTAT(1), IBYTE)
  INTEGER RECNUM, RAMERR
C
C *** CHECK TO SEE IF THE PICTURE IS TO BE SCALED
C
  ISCALV = 0
  IF(ISCALE .EQ. 1) GO TO 10
  IF(IMGROW .LE. 128) GO TO 20
  ISCALE = 2
  ISCALV = "401
  GO TO 10
20  ISCALE = 4
  ISCALV = "1002
C
C *** SET UP THE RAMTEK INSTRUCTION SET ***
C
10  IMBUF(1) = "5003
  IMBUF(2) = "2300
  IMBUF(3) = (640 - (IMGCOL*ISCALE))/2
  IMBUF(4) = (512 - (IMGROW*ISCALE))/2 + (RECNUM*ISCALE) - ISCALE
  IMBUF(5) = IMBUF(3) + (IMGCOL*ISCALE) - 1
  IMBUF(6) = IMBUF(4) + (ISCALE-1)
  IMBUF(7) = 0
  IMBUF(8) = ISCALV
  IMBUF(9) = 2*IMGCOL
C
C *** CHECK IF INITIAL WRITE ***
C
  CALL GETADR(IPAR, IMBUF)
  IPAR(2) = IMBUF(9) + 18
  IPAR(3) = 0
C
C *** WRITE OUT THE IMAGE ***
C
  CALL WTQIO("410, LUN, 2, 3, IOSTAT, IPAR, RAMERR)
  IF(RAMERR.EQ.1)RAMERR = 0
  IF (RAMERR .GE. 0) RETURN
C

```

C *** WRITE RANTEK ERROR MESSAGE ***
C
CALL RMIOER(18YTE,RAMERR)
RAMERR = 1
C
RETURN
END

```

C      SUBROUTINE IMDSP2(LUNRM,IMDERR)
C
C      THIS SUBROUTINE WRITES AN IMAGE TO THE RAMTEK DRIVER, THEN
C      OVERLAYS THE INTERESTING POINTS OF AN INTERESTING POINT SET
C      ON THE DISPLAYED IMAGE.
C
C      INCLUDE 'COMMON.FTN'
C      COMMON /IFILE1/ LUNIPS
C      COMMON /CWFILE/ LUNCW
C      COMMON /IPS/ IPKNT,IPR(M),IPC(M)
D      WRITE (6,*) '*** ENTER IMDSP2 ***'
C      LUNCW = LUNRM
C
C      *** DISPLAY THE SELECTED IMAGE USING THE RAMTEK DRIVER ***
C
C      CALL IMDSP1(LUNRM,ISCALE,IMDERR)
C      IF(IMDERR.EQ.1)RETURN
C
C      TYPE *, ' '
C      TYPE *, 'ENTER THE LOGICAL UNIT NUMBER AND FILE NAME OF THE'
C      TYPE *, 'SET OF INTERESTING POINTS YOU WISH TO DISPLAY.'
C      CALL IF1FLU
C
C      *** OPEN THE INTERESTING POINT SET FILE ***
C
C      CALL OPNIF1
C
C      *** READ THE NUMBER OF INTERESTING POINTS ***
C
C      READ (LUNIPS,1) IPKNT
I      FORMAT(I3)
C
C      *** INPUT THE INTERESTING POINT LOCATIONS ***
C
C      DO 10 I=1,IPKNT
C      READ (LUNIPS,2) IPROW,IPCOL
2      FORMAT(2I3)
C      IPR(I) = IPROW
C      IPC(I) = IPCOL
10     CONTINUE
C
C      *** WRITE THE INTERESTING POINTS ON THE RAMTEK IMAGE DISPLAY ***
C
C      CALL RAHIP(LUNRM,ISCALE,IMDERR)
C
C      *** CLOSE THE INTERESTING POINT SET FILE ***
C
C      CLOSE(UNIT=LUNIPS)
C

```


RETURN
END

```

SUBROUTINE RAMIP(LUN, ISCALE, RAMERR)

C
C THIS SUBROUTINE WRITES THE INTERESTING POINTS ON THE IMAGE
C PREVIOUSLY DISPLAYED BY SUBROUTINE RAMWT.
C

INCLUDE 'COMMON.FTN'
COMMON /IPS/ IPKNT, IPR(M), IPC(M)
COMMON /RBUF/ IPAR(6), IOSTAT(2)
COMMON /IMAGE/ IMGROW, IMGCOL
DIMENSION IMBUF(512)
LOGICAL * 1 IBYTE
EQUIVALENCE (IOSTAT(1), IBYTE)
INTEGER RAMERR
DO 10 I = 1, IPKNT

C
C *** SET UP THE RAMTEK INSTRUCTION SET ***
C

IMBUF(1) = "5003
IMBUF(2) = "300
IMBUF(3) = (640-(IMGCOL*ISCALE))/2 + (IPC(I)-1)*ISCALE-1
IMBUF(4) = (512-(IMGROW*ISCALE))/2 + (IPR(I)-1)*ISCALE-1
IMBUF(5) = IMBUF(3) + 2
IMBUF(6) = IMBUF(4) + 2
IMBUF(7) = 0
IMBUF(8) = 18
DO 20 J=1,9
IMBUF(8+J) = 256
20 CONTINUE
C
C *** SET UP WRITE STREAM ***
C

CALL GETADR(IPAR, IMBUF)
IPAR(2) = IMBUF(8) + 16
IPAR(3) = 0

C
C *** WRITE OUT THE IMAGE ***
C

CALL WTQIO("410, LUN, 2, 3, IOSTAT, IPAR, RAMERR)
IF (RAMERR .EQ. 1) RAMERR = 0
IF (RAMERR .LT. 0) GO TO 30
10 CONTINUE
RETURN

C
C *** WRITE RAMTEK ERROR MESSAGE ***
C

30 CALL RMIOER(IBYTE, RAMERR)
RAMERR = 1

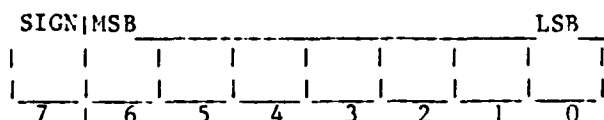
C
RETURN

```

END

Appendix G: Byte-to-Word Number Conversion

When an 8-bit number is read by the computer (in this case, a DEC PDP 11/45 minicomputer operating under the RSX-11M operating system) and stored in a byte storage location, it is stored as shown below:



If the sign bit is a "1", the number is considered to be negative. If it is "0" (zero), the number is positive. Thus, positive numbers up to 01111111 = 127, and negative numbers down to 10000000 = 128, can be stored. To make the numbers negative (say -1), it is necessary to add the 2's complement to the number to be negated. For example, for +1 (represented as 00000001), the corresponding negative number (-1) is not represented as 10000001, as might be expected, but as the 2's complement of 00000001. This formed by inverting all the bit positions (0->1, 1->0) and adding 1. Thus, -1 = 11111110 + 1 = 11111111, -2 = 11111110, -3 = 11111101, and so forth.

Now consider the manner in which the intensity data is input. For numbers between 0 and +127, there is no error. However, the number +128 is input as 10000000. The computer interprets this not as the smallest negative number (-1), but the largest (-128). Now consider the largest positive number which can be input (255). When attempting to store this number, it is represented as 11111111 on the image file (which is the binary representation of 255). As mentioned above, the computer, since it

also considers possible negative number inputs, interprets 11111111 as -1, the smallest negative number. Thus, the conversion process works backward from what might be expected. If the number input is 255, it will be internally stored in the computer as -1. Therefore, 256 must be added to the value to obtain the correct number. Similarly, 254 will be stored as -2. Again, adding 256 restores the correct value.

The way this conversion process works internally is that the number is stored in a 16-bit integer word when equivalenced with the byte array value. Thus, the number 255 (stored as -1 = 11111111 in the byte location) becomes 1111111111111111, the 16-bit representation of -1, when equivalenced to a 16-bit integer word. When the 16-bit number 256 is added to this, the following results:

$$\begin{array}{r}
 1111111111111111 \quad (-1) \\
 +1000000001000000 \quad (+256) \\
 \hline
 1100000001111111 \quad (+255)
 \end{array}$$

Since the 1 at the leftmost position is lost, the 16-bit numerical result is 255, the desired number. Thus, the conversion algorithm is quite simple:

1. Equivalence the 8-bit byte array to a 16-bit word array after the intensity values are read.
2. Test the 16-bit word to see if it is less than zero.
3. If the 16-bit word is less than zero, add 256 to it.
4. If the 16-bit word is greater than or equal to zero, leave it as is.

Appendix H: NVL Data Base Tape-to-Disk Conversion Program

This appendix contains the FORTRAN source program used to convert the NVL data base to a disk file acceptable for input to program DIDA.

```

      BYTE MTST
      LOGICAL*1 IANSW
      INTEGER*2 MTSTAT(2),MTPARM(6)
      BYTE MTRBUF(4500)
      EQUIVALENCE (MTST,MTSTAT)
      IFLAG = 0
      ICNT = 1
      TYPE 1000
      ACCEPT 2000, IANSW
      TYPE 1010
      ACCEPT 2010, NUMREC
      CALL ASNLUN(1,'MT:',0)
      CALL ASNLUN(2,'SY:',0)
      CALL QIO ("1400,1,,MTST,,MTDS)
12      IF(MTST)11,12,13
   C
   C      1 MEANS SUCCESSFUL COMPLETION
   C      0 MEANS OPERATION STILL PENDING
   C      -1 MEANS UNSUCCESSFUL COMPLETION
   C
11      CONTINUE
13      CONTINUE
      WRITE(5,101)MTST,MTSTAT(2)
101     FORMAT('ATT',2I5)
      CALL WTQIO("2400,1,1,,MTST)
      CALL WTQIO("2500,1,1,,MTST,0)
17      CONTINUE
   C
   C      CHECK REWIND BIT TO SEE IF TAPE IS REWOUND
   C      2520 = SENSE TAPE CHARACTERISTICS
   C
      CALL WTQIO("2520,1,1,,MTST,IU)
      IF(IAND(IU,"1000).NE.0) GO TO 17
   C
   C      FIND ADDRESS OF BUF TO SEND TO
   C      MAGTAPE DRIVER
   C
      CALL GETADR(MTPARM,MTRBUF)
      MTPARM(2)=4500
5      IASV=1
      OPEN(UNIT=2,NAME='DK:[10,20]TERAIN.IMG',FORM='UNFORMATTED',
1       TYPE='NEW',ACCESS='DIRECT',RECORDSIZE=128,MAXREC=512,
```

```

2    ASSOCIATEVARIABLE=IASV)
    IF(NUMREC.NE.1)CALL WTQIO("1000,1,1,,MTST,MTPARM')
C
C    READ A LOGICAL BLOCK    (OCTAL CODE = 1000)
C
10   CALL    WTQIO("1000,1,1,,MTST,MTPARM)
    IF(MTST.NE.1)TYPE*, 'IOSTATUS = ',MTST
    IF(MTST.NE.1)GO TO 300
    IF(IANSW.EQ.'Y')TYPE*, ' BYTES READ = ',MTSTAT(2)
    IFLAG=0
    DO 16 M=2,4096,2
    IF(MTBUF(M).GE.0)GO TO 14
    MTBUF(M)=MTBUF(M)/2
    MTBUF(M)=MTBUF(M)-128
    GO TO 20
14   MTBUF(M)=MTBUF(M)/2
20   IF(MTBUF(M-1).EQ.1)MTBUF(M)=MTBUF(M)-128
16   CONTINUE
    DO 15 I=1,NUMREC
    WRITE(2'IASV) (MTBUF(I1),I1=1024*I-1022,1024*I,2)
15   CONTINUE
    GO TO 10
300  IFLAG = IFLAG +1
C
C    AFTER EACH END OF FILE CLOSE CURRENT OUPUT FILE AND OPEN ANOTHR ON
C
C
    CLOSE(UNIT=2)
    TYPE*, 'EOF', ICNT
    IF(ICNT.EQ.20)GO TO 999
    IF(IFLAG.EQ.2) GO TO 999
    ICNT = ICNT +1
    GO TO 5
999  STOP
1000 FORMAT(1X, 'DO YOU WANT BYTES READ PRINTED? [Y/N] 'S)
1010 FORMAT(1X, 'ENTER NUMBER OF RECORDS (1 OR 4) '/'    NUMREC = ',S)
1100 FORMAT(80A1)
2000 FORMAT(A1)
2010 FORMAT(I3)
    END

```

VITA

Franklin D. Cooper III was born on 4 December 1944 in Princeton, Indiana. He graduated from F.J. Reitz High School in Evansville, Indiana in 1962. He entered the United States Air Force in 1964 and served at enlisted rank as an Instrument Trainer Technician. He received the degree of Bachelor of Science in Electrical Engineering in December 1976 from Texas Tech University, Lubbock, Texas through the Airman Education and Commissioning Program (AECPP). After being commissioned in May 1977, he served as an R&D engineer at the Air Force Human Resources Laboratory (AFHRL) at Wright-Patterson Air Force Base, Ohio. During his stay at AFHRL, he performed research and development in the area of flight and maintenance training simulation. He entered the School of Engineering, Air Force Institute of Technology, in June 1980.

Permanent address: 1626 Fleener Road
Evansville, Indiana 47711

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GEO/EE/81D-1	2. GOVT ACCESSION NO. A127409	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DISPARITY ANALYSIS OF TIME-VARYING IMAGERY		5. TYPE OF REPORT & PERIOD COVERED MS Thesis
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Franklin D. Cooper III, Captain, USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN)		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Information Processing Technology Branch Air Force Avionics Laboratory (AFWAL/AAAT-1) Wright-Patterson AFB, OH 45433		12. REPORT DATE December 1981
		13. NUMBER OF PAGES 369
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES Approved for public release: LAW AFB 180-17. E. E. WOLVER Dean for Research and Professional Development Air Force Institute of Technology (AIC) Wright-Patterson AFB OH 45433 7 APR 1982		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Disparity Analysis Matching Algorithms Correspondence Problem Time-Varying Imagery Interest Operators Dynamic Imagery Image Processing Interesting Points		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) An interactive program (called program DIDA) was developed to perform disparity analysis of time-varying imagery. The program consists of four sections: (1) image operations; (2) interesting point selection; (3) interesting point matching; and (4) image display. Three data bases were obtained: (1) a 20-frame sequence of images from a terrain board of the U.S. Army Night Vision Laboratory; (2) an 18-frame sequence of various moving objects from the University of Minnesota; and (3) a 33-frame sequence of images of a traffic scene from the University of Hamburg.		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

— Each of the four program sections were tested. Image operations included printing image intensities, and image file creation and deletion. Three interest operators were employed in the interesting point selection section: (1) simple variance; (2) directed variance (the Moravec Operator); and (3) edged variance. The number of interesting points selected was found to be inversely proportional to the interest operator threshold. A relaxation-labeling matching algorithm was used in the interesting point matching section to match interesting points from two images. Very poor matching results were obtained. Image displays included black-and-white and color images with and without the interesting points overwritten. Photographs were taken of both cases.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

DAT
ILM