AIM

Association for Information and Image Management

1100 Wayne Avenue, Suite 1100
Silver Spring, Maryland 20910
301/587-8202

Centimeter

Inches

MANUFACTURED TO AIM STANDARDS
BY APPLIED IMAGE, INC.

1 of 1

EGG-EE-11298

**Automatic TLI Recognition System,
Part 2: User's Guide**

**J. K. Partin
G. D. Lassahn
J. R. Davidson**

**HQ PROJECT MANAGER - Stephen Herrick
PROJECT NUMBER - ST474E**

Published May 1994

**Idaho National Engineering Laboratory
EG&G Idaho, Inc.
Idaho Falls, Idaho 83415**

**Prepared for the
U.S. Department of Energy
Office of Arms Control
under DOE Idaho Field Office
Contract DE-AC07-76ID01570**

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

ABSTRACT

This volume is a user's manual for an Automatic Target Recognition (ATR) system. General information about this ATR system is contained in EGG-EE-11298, *Automatic TLI Recognition System, Part 1: System Description*.

CONTENTS

INTRODUCTION	1
SOFTWARE DESCRIPTION	2
Kernels and Excluded Edge Regions	2
Overlap Rows	3
Image Compatibility	9
User-Accessible Variables	9
Training Program F	10
Surveillance Program E	12
SCSI Software	13
PROGRAM USAGE	14
Mask Creation Program G	14
Training Program F	16
Surveillance Program E	18
SCSI Software	19
APPENDIX A	
EXAMPLE 1: Median Filter Testing	A-1
APPENDIX B	
EXAMPLE 2: Roads and Riverbanks	B-1
APPENDIX C	
EXAMPLE 3: Buried Waste Location	C-1
APPENDIX D	
EXAMPLE 4: Finding Airplanes	D-1
APPENDIX E	
E9 AND F9 FEATURE CALCULATION COMMANDS	E-1

Automatic TLI Recognition System, Part 2: User's Guide

INTRODUCTION

This is Part 2 of a two-part report on an automatic target recognition (ATR) system. This guide is intended to provide enough information and instruction to allow individuals to use the Automatic Target Recognition (ATR) system for their own applications. It is assumed here that the potential user has Part 1 of this report, which contains basic descriptions and some of the figures referred to in the appendices of this Part 2. This report describes version 9 of the E and F programs, which run on a T805 transputer network.

SOFTWARE DESCRIPTION

The user must be aware of a few basic concepts and limitations inherent in this ATR system. These are described here.

Kernels and Excluded Edge Regions

A common image analysis operation is convolution of the image with a small "kernel" or "operator". A kernel is like a small image, in that it is a two-dimensional array of values, or pixels. This convolution operation is defined by $D_{m,n} = \sum S_{m+i,n+j} K_{i,j}$, where $D_{m,n}$ is the row m , column n pixel in the destination image, $S_{m,n}$ is the m,n pixel in the source image, $K_{i,j}$ is the i,j pixel in the kernel, and the sum is over all values of i and j for which $K_{i,j}$ is defined. (This is not strictly a convolution calculation, which would require subtracting instead of adding to get the indices of S .) Intuitively, this can be thought of as positioning the kernel so that its $i=0,j=0$ pixel coincides with the m,n pixel of S , summing the products of the S and K pixel values for all the pixels in K , and assigning this sum value to the m,n pixel in D . Usually, the $i=0,j=0$ pixel is in the center of the kernel, but this is not always the case. In fact, the $i=0,j=0$ pixel may not even be part of the kernel. The kernel could be defined, for example, for $-3 \leq i \leq 2$ and $1 \leq j \leq 4$, so that $i=0,j=0$ is not part of the kernel's domain and there would be no $i=0,j=0$ term in the sum.

This definition of convolution is not valid for edge pixels of the destination image D , for values of m,n such that $S_{m+i,n+j}$ is not defined for some of the i,j values for which $K_{i,j}$ is defined. We refer to these edge pixels of D , for which the kernel K would extend past the edge of the source image S , as *excluded edge* pixels. These pixel values are not defined by the convolution operation. For the example kernel domain just mentioned, the excluded edge region would comprise the leftmost 3 columns of pixels (for $i_{\min}=-3$ which is less than 0), the rightmost 2 columns (for $i_{\max}=2$ which is greater than 0), and the bottom 4 rows (for $j_{\max}=4$ which is greater than 0; j is positive downward); no rows would be excluded at the top, because $j_{\min}=1$ is not less than 0 so no part of the kernel will ever extend past the top of the source image. This concept of excluded edge pixels is not limited to convolution calculations, of course. It is equally applicable to any operation involving a kernel or a similarly-defined local region that must be contained entirely within the source image to produce a well-defined result for the destination image. Examples of operations in this ATR system which result in excluded edge regions are convolution (CONVOLVE), median filtering (MED1X, MED1Y, MEDIAN), local sum of squares (SSQ), quadratic polynomial fitting (QUADXY and QUADUV), and gradient calculations (GRADT, XGRAD, YGRAD). Perhaps contrary to one's expectation, NTRP00, NTRP01 do not produce excluded edge regions; these operations do not require that there be a source image pixel for every kernel pixel.

Often, when an operation like convolution forms an image with undefined pixel values at the edges, we would like to exclude those edge pixels from later operations. Some operations, such as SCALE, allow the user to exclude these edge pixels by telling the operation which kernel produced, or would have produced, the undefined edge region. Kernel 0 is always defined to have a one-pixel domain, with $i_{\min}=i_{\max}=j_{\min}=j_{\max}=0$, so that specifying kernel 0 in this context implies that no pixels are excluded. It is permissible to define a kernel of the desired size solely

for the purpose of specifying an excluded edge region, without ever using the kernel for any other purpose.

Overlap Rows

Parallel computing is accomplished in this system by dividing each image among several nodes. Each node is a computer which has its own memory and can do computations on the data in its own memory. In the simplest cases, the images could be distributed among the nodes in any manner, as long as all the images are distributed in the same way. One approach is to treat each row of pixels as one unit, and assign different rows to different nodes. This approach is illustrated in the Table 1 example, for which each image comprises 19 rows (with some unspecified number of columns) of pixels which are distributed among 5 slave nodes. This simple approach serves very well for calculations such as adding two images, in which the calculations for one pixel do not involve any neighboring pixel's values. However, for some calculations, such as smoothing, convolution, and median filtering, the result for one pixel requires input of values of neighboring pixels. For some of the pixels, the required neighboring pixels will be in different nodes. This problem could be handled by having the nodes request the necessary pixel values from other nodes as required. Or, as is done here, we could have each node store the necessary neighboring pixel values in its own memory along with its "own" pixel values. In this system, each node may store one or more extra rows of pixels, which we call *overlap* rows, in addition to its own "*primary*" rows. The values in these overlap rows do need to be updated sometimes, by requesting current values from the node in which these rows are the primary rows. However, this updating of overlap row values does not need to be done very frequently, and it is more efficient to exchange a previously specified block of data (all the overlap rows) in one operation than to exchange each neighboring pixel value as required in a separate operation. Tables 2 and 3 illustrate how an image is distributed among several nodes with 1 and 2 overlap rows respectively. In the Table 2 example, before doing an operation that required nearest neighbor pixel values, node 2 would request the current values of the row 4 pixels from node 1 and the row 9 pixel values from node 3.

**Table 1: Distribution of
19 Image Rows among 5 Nodes
with 0 Overlap Rows**

NODE =	1	2	3	4	5
IMAGE ROW:					
0	x				
1	x				
2	x				
3	x				
4		x			
5		x			
6		x			
7		x			
8			x		
9			x		
10			x		
11			x		
12				x	
13				x	
14				x	
15				x	
16					x
17					x
18					x

An "x" in a row and column indicates that the corresponding image row is in the node with that column number. Thus, image rows 0 through 3 are in node 1; rows 4 through 7 are in node 2; etc. The different nodes may have different numbers of image rows.

**Table 2: Distribution of
19 Image Rows among 5 Nodes
with 1 Overlap Row**

NODE =	1	2	3	4	5
IMAGE ROW:					
0	x				
1	x				
2	x				
3	x				
4	x	o			
5	o	x			
6		x			
7		x			
8		x	o		
9		o	x		
10			x		
11			x	o	
12			o	x	
13				x	
14				x	o
15				o	x
16					x
17					x
18					x

An "o" in the table indicates that the node has space for, and sometimes actually holds data for, that image row which is referred to as an overlap row. Thus, node 1 has space for image rows 0 through 5; it always holds the current values for rows 0 through 4, and it may or may not hold the current values for row 5, the overlap row. In node 2, row 4 and row 9 are overlap rows.

Table 3: Distribution of
19 Image Rows among 5 Nodes
with 2 Overlap Rows

NODE =	1	2	3	4	5
IMAGE ROW:					
0	x rlo,slo				
1	x				
2	x				
3	x	o rlo			
4	x shi	o			
5	o	x slo			
6	o rhi	x	o rlo		
7		x shi	o		
8		o	x slo		
9		o rhi	x	o rlo	
10			x shi	o	
11			o	x slo	
12			o rhi	x	o rlo
13				x shi	o
14				o	x slo
15				o rhi	x
16					x
17					x
18=nrow-1					x rhi,shi

The slave node receives all primary and overlap rows, rows **rlo** through **rhi**, from the master when an image is loaded into memory. The slave node sends only its primary rows, rows **slo** through **shi**, when an image is copied from memory to a disk file. **nrow** is the number of rows in the whole image, 19 in this example.

**Table 4: Alternate Distribution of
19 Image Rows among 5 Nodes
with No Overlap Rows**

NODE =	1	2	3	4	5
IMAGE					
ROW:					
0	x				
1	x				
2	x				
3	x				
4	x				
5		x			
6		x			
7		x			
8			x		
9			x		
10			x		
11				x	
12				x	
13				x	
14					x
15					x
16					x
17					x
18					x

**Table 5: Alternate Distribution of
19 Image Rows among 5 Nodes
with 2 Overlap Rows**

NODE =	1	2	3	4	5
IMAGE ROW:					
0	x				
1	x				
2	x	o			
3	x	o			
4	o	x			
5	o	x			
6		x	o		
7		x	o		
8		o	x		
9		o	x		
10			x	o	
11			x	o	
12			o	x	
13			o	x	
14				x	o
15				x	o
16				o	x
17				o	x
18					x

This is a possible alternate distribution which optimizes calculation efficiency at the expense of storage space efficiency, as compared with the Table 3 example. This configuration would not occur via a simple image definition command in this system, but it could be created by an image definition copy command by starting with the Table 1 example and specifying an image copy with 2 overlap rows in the destination. Calculations with this distribution would typically take 4/5 as long as the same calculations on the Table 3 distribution. This distribution in effect occupies 8/7 as much memory as the Table 3 distribution.

Operations such as CONVOLVE require that the appropriate overlap rows be present and that the pixels in these overlap rows are set to the correct current values. The number of overlap rows must be enough to accommodate the kernel being used, so that the kernel never extends beyond the overlap rows of any node when the $i=0, j=0$ pixel of the kernel is in a primary row. That is, the number of overlap rows should be at least as large as the larger of $j_{\max}$ or $-j_{\min}$ for the kernel being used. The presence of overlap rows can be accomplished by specifying an appropriate value for the number of overlap rows (usually called `novl`) when defining an image using commands like DEFIMG and RESAMPLE. Ensuring that the overlap rows have the correct values requires a little more care. Most operations (ADD, CONVOLVE, RESAMPLE, and many others) do not set the values of the overlap rows. That is, when we add two images, for example, the overlap rows of the destination image will not be set to the current, correct values. The overlap row values are set only by the operations READIMAGE, READSCENE, ZEROIMAGE, INSERT, EXTRACT, and OVERLAP. The user must be sure that one of these four operations (normally OVERLAP) is done on each image requiring overlap rows, before the operation that requires the overlap rows and after any previous operation that changes primary rows without changing overlap rows.

Image Compatibility

For two images to be compatible for most image operations, such as adding two images or smoothing an image, the two images must be the same size, having the same number of rows and the same number of columns, and their primary rows (not their overlap rows) must be distributed in the same way among the several nodes. The two images represented in Tables 1 and 5 are compatible, and the two in Tables 3 and 4 are compatible; all other image pairs from Tables 1 through 5 are incompatible. We cannot add images 1 and 2, nor can we use image 2 as a destination for an addition operation that uses image 1 as one of the sources. In this system, the user does not have much detailed control over how an image is distributed among the nodes. However, the user can be sure that a new image that he defines is compatible with an already-defined image by copying that image definition using the COPDEF command. In this copying process, the number of overlap rows may be different (at the user's discretion) without affecting compatibility. Image 4 could be formed by copying the definition of image 3, or vice versa. More generally, any two images that are the same size and were defined by the same command (COPDEF, READIMAGE, READSCENE, RESAMPLE using the same source image, or UNDERSAMPLE using the same source image) are compatible for two-image operations.

User-Accessible Variables

Programs E and F (described later) allow the user to define 32-bit integer and 32-bit floating point scalar variables (not arrays), print their values, and use them as parameters in feature calculation commands. Some user-accessible variables are pre-defined and their values are automatically set by certain operations. The pre-defined user-accessible variables are named \$N, \$NCOL, \$NROW (integers), and \$AVG, \$SIG, \$MAX, \$MIN, \$A, and \$B (floating). The programs do not distinguish between upper and lower case characters in variable names. The arithmetic operations defined for these variables allow mixing of integer and floating types.

Training Program F

The training processes is implemented in program F. The training program F analyzes the training scene images and masks, calculates feature images according to an operator-supplied list of instructions, and finds the optimum values of coefficients to be used with these features in the surveillance program E. There are three main parts of the program F: feature calculation, sum calculation, and coefficient calculation.

Feature Calculation

Feature calculation typically starts with a single scene image and calculates several feature images. This may involve arithmetic or logical operations on a pixel-by-pixel basis; convolution in two dimensions with a template that is typically much smaller than the scene image; morphological operations; local order sorting operations, such as are done in a median filter; local co-occurrence matrix calculation; and others, limited only by the imagination of the operator. Of course, a given computer program can implement only a finite number of operations. The operations currently in this F program are listed in Appendix D.

The final step in the calculation of a single feature is writing the feature image to a disk file. Of course, it would be much faster to store all the feature images in memory. However, image storage requires a lot of memory, and this in-memory storage approach would require that the number of features be limited according to the amount of memory available. To make the program as general as possible and avoid unnecessary restrictions on the number of features, this approach of writing the feature images to disk was adopted. Feature images are not written automatically; the user must write each feature image, preferably with the WRITEFEAT command. In some cases, it is not necessary to write a feature image to a disk. If the desired feature image already exists in a disk file, the user can simply enter the name of that file into the F program's internal list of feature image files, using the FEATFIL command. If a feature is the product of two previously defined features, the product feature image does not need to exist as a disk file, and the user can simply enter a product code into the F program's internal list of feature image files, using the FEAT* command.

Because of the format with which the feature images are written, each feature image pixel value must be between 0 and 255. This imposes a restriction, that the feature images must be non-negative. The other apparent restriction, requiring values less than 255, is easily handled by simply using a SCALE command immediately before writing each feature image to disk. This scales the image to the acceptable range and includes the scale factor in the disk file, so that when the feature image is later read from the disk file it is automatically rescaled to the original values.

Along with the real feature images for each training scene, there must be a mask. This mask must contain the same information that is in the mask output by the mask creation program G, and it must be of the same size as the feature images. If the feature images are the same size as the scene images, the same file output by the mask creation program G will serve as the mask for these feature images, and the FEATFIL command is useful for telling the F program to use that file as the feature image mask. On the other hand, if the feature images are not the same size as the scene images, a new mask of the correct size must be created for the feature images. This can normally be done very easily by simply resampling the scene mask that was output by

the mask creation program G, using RESAMPLE or UNDERSAMPLE, and writing the resampled image using WRITEFEAT. The mask for the feature images must always be the first feature image (number 0), whether it is written using WRITEFEAT or referenced using FEATFIL. The mask should not be scaled (using SCALE) before it is written.

Sums Calculation

This ATR training process in the F program assumes (with some justification) that the distributions of result image pixel values are approximately Gaussian, both for the target pixels and for the background pixels. One result of this is that the error minimization process requires sums of pixel values from every product of two feature images. It would be more convenient to calculate these products and sums if all the feature images could be in memory at one time, but this would limit the number of features depending on memory availability. The approach used here is to read one row of pixels from every feature image into memory, and calculate the products and accumulate the sums from that row before replacing it with the next row from all feature images. This means that the program must repeatedly access a series of disk files, which is somewhat time consuming, but this seems to be the best alternative.

Program F automatically does the summing after calculating and writing the feature images. After the sums are accumulated, the feature images are no longer needed. One result of this is that successive training scenes can all use the same names for feature image disk files without conflict, unless the user for his own reasons wants to preserve all the feature images for the several training scenes. The sums are all that is required to complete the training procedure. In addition to being used in the last part of the F program, the sums are written to a disk file to save the effort of re-accumulating them if they are needed in later runs of F.

Coefficient Calculation

After the sums are calculated for a given set of training data, program F calculates the optimum values of the coefficients for the full set of features and for certain feature subsets. It will often happen that a substantially smaller subset will give error rates that are essentially the same as the error rate for the full original feature set. It is important to know this, because efficiency of the surveillance process (the analysis of images in the field) is greater for smaller feature sets. Therefore, the F program is arranged to look for the best feature subsets. For each case, F calculates the optimal coefficient value for each included feature, and F also calculates the importance of keeping each feature included in that case. Then, a new case is formed by deleting the least important feature from the old set. The results for each case are written to the F output file. This process of forming successively smaller subsets by deleting one feature at a time does, in effect, yield a graph of total error rate versus number of features, like the example in Figure 4. The user can then see how many features he must include in the surveillance process to attain the desired error rate, and he can judge the trade-off between increased error rate and increased number of features.

Figure 4: Example of total (weighted sum) error rate versus number of features. The optimization calculation tried several times, with slightly different resulting error rates, for some numbers of features; this is apparent from the double points plotted for 36 and 37 features.

For each case, the optimum coefficient values are determined by minimizing the total error rate. Program F does this minimization by using a linear approximation of the non-linear equations. This is an iterative, successive approximation procedure, and it requires a first guess for the coefficient values. A first guess can be obtained from a simple linear least squares fitting process; or, in the F program, a first guess can sometimes be obtained from the results of a previously calculated case which includes most of the same features as the present case. As is usually the case in this type of calculation, it is possible that the process will not find the desired absolute minimum in the error rate, but will converge to a substantially different relative minimum point. Sometimes this is evidenced by a smaller subset of included features giving a smaller error rate than a larger, previously calculated feature set. When this occurs, program F uses the coefficient values from the smaller subset, smaller error rate case as a first guess and restarts with the full original set of features included. This procedure sometimes leads to the discovery of optimal points with error rate minima that are substantially smaller than those associated with the first-indicated optimal point for a given number of features.

Surveillance Program E

The surveillance program E is used in the field to analyze a set of scene images (no mask) and report on whether there is a target present in the scene. Program E uses the values of the coefficients determined by program F. Program E inputs scene images and outputs a result image. If the value of any pixel in the result image is greater than Q, a value supplied by the training program F, that pixel is interpreted as an indication of the presence of a target. The result image can be written to a disk file with the Q value embedded in the file so that the image can be displayed later with obvious indications of any targets that may be present.

The feature calculation file for program E must have built into it the values of the coefficients determined by a previous run of the training program F. The program E feature calculation file must specify calculation of the required feature images as they were calculated for the F program, and it must multiply each of these feature images by the appropriate coefficient from the F program output (MULCON FEATURE FEATURE COEFn, for example, for feature $n > 0$). All of these feature images must then be added to produce a result image (ADD RESULT RESULT FEATURE for each feature). (Of course the result image must be zeroed initially, ZEROIMAGE RESULT). Usually, the constant feature image, $f=0$, must also be added to the result image. This is done by adding to the result image a constant, equal to the feature number 0 coefficient from the F program output (ADDCON RESULT RESULT COEF0). For output and display purposes, the result image should be clipped, so that there are no values greater than 255 or less than 0 (MINCON RESULT RESULT 255.0 and MAXCON RESULT RESULT 0.0). If the user wishes, he can write the result image to a disk file with the value of Q in the file in place of the value of the scale factor A, for later display or analysis (WRITERESULT RESULT 1 Q).

SCSI Software

The SCSI bus is interfaced directly to the transputer network, not to the host computer. One result of this is that the SCSI devices (disk drive and tape drive) are not controlled by the host DOS operating system, and the familiar DOS disk file system is not used on the SCSI devices. In this application, separate file handling system software is used for the SCSI devices. The SCSI disk may be used by programs E and F very much as if it were a DOS disk with the device name SCSIDISK: (instead of a usual DOS disk name like A: or C:). One major difference is that SCSIDISK file names are case sensitive: upper case and lower case letters are treated as distinct for the SCSIDISK, contrary to the DOS file system. A second difference is that the SCSI disk does not have sub-directories; there is only one directory, and it can list only 512 files. The SCSI disk file names can include the characters ":", "\", and ".", so that SCSI disk file names can look like DOS file names with device and path information, and in some ways the SCSI disk file names can be treated as if they were DOS multi-level directory path/file names. SCSI disk file names can not be longer than 52 characters. The SCSI tape drive is not accessible via programs E or F.

A separate program, TSCSI, was written to allow more specific access to the SCSI devices. This program allows such operations as copying files to and from the SCSI devices, reading the disk directory, deleting files, and certain diagnostic operations.

PROGRAM USAGE

Mask Creation Program G

The mask creation program G is executed by typing "run G". Program G accepts two types of keyboard input: word commands, with parameters, while not in the image display mode; and, single-key-stroke commands, while in the image display mode. The position of the cursor, row and column, is displayed on the console screen during the display mode; the top left pixel is row 0, column 0.

Word Commands

DEFIMG ncol nrow

This tells the program the size of the images you will be using. **ncol** is the number of columns, the width of the image in pixels, and **nrow** is the number of rows, the height of the image in pixels. This command should usually be used before any other command, and this command must be used again if you change from working with one size to a different size of image. To use this command, type "defimg", followed by a space, followed by the numerical value of **ncol**, followed by a space, followed by the numerical value of **nrow**, followed by the Enter or Carriage Return key. The word "defimg" may be in upper case or lower case.

INS filename row0 col0 ncols header bpp

This tells the program to get a scene image from a disk file. This command uses the same subroutine as the READIMAGE command of the E and F programs, but the parameters are (obviously) listed in different orders for the INS command and the READIMAGE command. For the INS command, if the last 5 or fewer parameters are missing from the command line, the values used in the previous INS command are used again here. Therefore, if you are working with a series of images of the same format, you can give values for the parameters **row0**, **col0**, **ncols**, **header**, and **bpp** for the first image input and not bother repeating these parameter values on later image input commands. If "." is given instead of a file name, the previous input file name is used again. If "+" is given instead of a file name, the previous input file name is incremented and then used here. After the scene image is input, the program automatically enters the image display mode.

INM filename row0 col0 ncols header bpp

This tells the program to get a mask image. The parameters and operation of this INM command are like those of the INS command.

MARK

This initiates the image display mode that allows marking target (region 2) and background (region 1) and unspecified (region 0, default) regions on the scene image.

OUTS filename

This copies the scene image from memory to the disk file **filename**, with a header. The "." and "+" codes can be used to repeat the previous scene image output file name.

OUTM filename

This copies the mask from memory to the disk file **filename**, with a header. The "." and "+" codes can be used to repeat the previous mask output file name.

CLR

This destroys any stored image definition information.

STOP

This stops execution of the mask creation program G.

*Single-Key-Stroke Commands***cursor keys**

The cursor keys are used either with or without the shift key to move the cursor in steps of 25 pixels or 1 pixel, in the obvious manner. The numeric keypad cursor keys should be used.

, (comma)

The comma is a command to either start drawing a polygon at the cursor location or, if a polygon is already started, to continue the polygon with a line from the last-specified vertex to the cursor location.

. (period)

The period is a command to complete the polygon by drawing a line from the last-specified vertex to the first-specified vertex. This command does not move the cursor. You will then be asked to designate the type of region inside the polygon, 2 or 1 or 0. If you hit any other key, the mask will not be changed.

- (minus or dash)

The present polygon, which may be only partly complete, will be discarded.

; (semicolon)

The image and any existing polygon will be re-drawn. This may be desirable to remove the ashes of a discarded polygon.

! (exclamation point)

This is a command to leave the image display mode; it should be used when you have marked all the regions you want marked. The program will query you for a name for a mask output file; you may enter "nul" if you do not want the mask written to a disk file. The file name codes "." and "+" may be used here. This file output function is the same as that accomplished with the "OUTM" command.

Training Program F

The training program F is executed by typing "run F". This training program is controlled by a set of user-created ASCII files that contain instructions and data for the training processes. These files are described in the following paragraphs. The file name extensions used in these discussions are not required; they are used here to help clarify the descriptions.

Command File (extension .cmd)

When program F executes, it asks the operator for the name of a command file. The command file must contain the following information:

- line 1: an integer to set the operating mode. This value is not used in F, but it must be present in this file.
- line 2: the name of a scene list file (.sl) (input).
- line 3: the name of a feature calculation file (.fc) (input).
- line 4: the name of a sums file (.sum) (output).
- line 5: the name of a result file (.out) (output).
- line 6: W_1 , the relative weight for type 1 errors, a value between 0.0 and 1.0.

If an input file is not needed, or if an output file is not desired, the dummy file name "NUL" may be used on the appropriate line in this command file. F always requires a real scene list file (.sl). A feature calculation file (.fc) is not always needed, as indicated later. The operator always has the option of suppressing the generation of output files (.sum or .out) by using "NUL" as the file name.

Scene List File (.sl)

The program F gets the name of this scene list file from line 2 of the command (.cmd) file. In the most obvious mode of operation, this scene list file gives the number of training scenes, the number of images per scene, and the names of files containing the masks and images:

- line 1: number of scenes, an integer greater than 0.
- line 2: number of images per scene (not including mask).
- following lines, one set per scene:
 - line a: file name for the mask for this scene.
 - line b: file name for the first image for this scene.
 - following lines: file names for other images for this scene.

A feature command file (.fc) is required in this mode of operation.

In the second mode of operation, program F reads feature images rather than scene images, and a feature command file (.fc) is not required. This mode is indicated by representing the number of scenes by a negative number in this scene list file (.sl):

- line 1: negative of the number of scenes, integer less than 0.
- line 2: number of feature images per scene.
- following lines, one set per scene:
 - line a: file name for feature mask for this scene.
 - line b: file name for first feature image for this scene.
 - following lines: file names for other feature images for this scene.

In the third mode of operation, program F does not use any images. Instead, F simply

reads the sums that were calculated in a previous run of F. This mode is indicated by a value of 0 for the number of scenes in this scene list file (.sl):

line 1: the integer value 0.

line 2: the number of features (not including the feature mask or the unit feature in which all pixels have the value 1).

following lines: values of sums.

The sums output file (.sum) created by earlier runs of F can be used as the scene list input file (.sl) for this mode; it contains the correct information in the correct format. In this mode, a feature calculation file (.fc) is not required.

Feature Calculation File (.fc)

Program F gets the name of this feature calculation file from line 3 of the command (.cmd) file. This feature calculation file contains a list of key word commands which cause the generation of feature images from scene images. Each "instruction" in this feature calculation file comprises one line unless a file name is required, in which case the file name by itself comprises the second line of the instruction. The first line of each instruction is a key word command which indicates the operation to be done, followed by values for whatever parameters the command requires, in free format. Appendix D lists the commands. If the operation requires a file name or a coded alternative to a file name, it should appear by itself, with no remarks, on the second line of the instruction. A remark can be added to the first line, after all the parameter values are given. Such a remark must start with a semicolon ";". A remark can also appear on a line by itself, starting with either a semicolon or with the command "REM" or "ECHO". Separate remark lines must not be inserted between the two lines of a two-line command, a command which requires a file name on the second line.

Program F executes this sequence of instructions from this feature calculation file once for each scene, if the number of scenes given on line 1 of the scene list file (.sl) is greater than 0; otherwise, this feature calculation file is not used, and the dummy file name "NUL" may be used on line 3 of the command file (.cmd).

Sums File (.sum)

Program F generates a sums file (.sum) with the file name specified on line 4 of the command file (.cmd). The sums file is a set of numeric data that summarizes all the relevant information from the feature images and is used to calculate the optimal values of the coefficients to be used in the surveillance program E. The sums file would not normally be used except for one purpose: to execute F with the same set of features used in the F run that generated the sums file but with a different value for the error weight W_1 or with some variant of the F program. In such a repeated F run situation, it is much faster to simply read the sums file than to read all the scene images, calculate the feature images, and calculate the sums. As mentioned previously, the output sums file (.sum) from one run of program F can be used as the input scene list file (.sl) in later runs of F. The operator can suppress the generation of the sums file (.sum) by using "NUL" as the name of the sums file in line 4 of the command file (.cmd).

Result File (.out)

The essential results of program F are written to a result file. The data that is written for

each set of features is: the number of features, the error rates (type 1, type 2, and weighted sum), the value of the coefficient Q, the values of the coefficients C_i , and the importance for each feature in the set. The values of the coefficients are used by program E. The weighted sum error is a simple estimate of this case's uncertainty in target identification. It is an estimate of the fraction of training scene pixels that were incorrectly classified by program F during the training process. This result file normally includes results for several cases with different numbers of features. It may include several cases with the same number of features but with the included features being different in the different cases. It may even include two or more cases with exactly the same features but with different results, which different results may obtain from different first guesses at the coefficient values in calculating each case. (It is possible that the program may find different local minima instead of the desired absolute minimum in its search for the minimum error conditions for each case.) The operator must peruse the result file and decide which, if any, of the cases represented are best or acceptable for use with the E program. The operator can suppress the generation of the result file (.out) by using "NUL" as the name of the result file in line 5 of the command file (.cmd).

Surveillance Program E

The surveillance program E is started by typing "run E". Program E uses ASCII files similar to those used by F:

Command File (.cmd)

When program E executes, it asks the operator for the name of a command file. The command file must contain the following information:

- line 1: an integer to set the operating mode. This value is not used in E, but it must be present in this file.
- line 2: the name of a scene list file (.sl) (input).
- line 3: the name of a feature calculation file (.fc) (input).
- line 4: the name of an output file (.out) (output). Use "NUL" for this version of E.

E always requires both a scene list file (.sl) and a feature calculation file (.fc). The current version of the software does not write to the output file (.out); "NUL" should be given in line 4 of this command file (.cmd).

Scene List File (.sl)

The program E gets the name of this scene list file from line 2 of the command (.cmd) file. This scene list file gives the number of scenes to be analyzed, the number of images per scene, and the names of files containing the images:

- line 1: number of scenes, an integer greater than 0.
- line 2: number of images per scene.
- following lines, one set per scene:
 - line a: file name for result image to be output for this scene.
 - line b: file name for the first image for this scene.
 - following lines: file names for other images for this scene.

Feature Calculation File (.fc)

Program E always requires a feature calculation file, the name of which is specified in line 3 of the command file (.cmd). The feature calculation files for E are similar to those used for F, although the details of the content will be a little different. The E feature calculation file must contain instructions to read the scene images, calculate the required feature images, use the coefficients obtained from program F to multiply the feature images, add the feature images to create a result image, and display or write the result image. This feature calculation file is used once for each scene analyzed by E.

SCSI Software

The program TSCSI allows manipulation of the disk drive and the tape drive on the SCSI bus, which cannot be accessed via the host computer's DOS file handling system. This program is executed by typing "run TSCSI". It accepts commands from the keyboard. These commands can be typed in upper case or lower case, except that disk file names are case sensitive. In these commands, the parameter *id* is the SCSI device identification number, which in this system is 1 for the disk drive or 2 for the tape drive. If the SCSI system detects an error condition, an error message appears on the host computer monitor. The commands are as follows:

BUSRESET

SCSI BUS RESET command.

READY id

SCSI DEVICE READY command to device *id*.

REWIND id

SCSI REWIND command to device *id*.

REQSENSE id

SCSI REQUEST SENSE command to device *id*. Device response data appears on the host monitor in hexadecimal format.

INQUIRY id

SCSI INQUIRY command to device *id*. Device response data appears on the host monitor in hexadecimal format.

MODESENSE id

SCSI MODE SENSE command to device *id*. Device response data appears on the host monitor in hexadecimal format.

CAPACITY id

SCSI READ CAPACITY command to device *id*. Device response data appears on the host monitor with capacity values in hexadecimal format.

INIT id

Initiate device **id**. For the disk, this command will DELETE ALL DATA and initialize the directory and the file allocation table. For the tape drive, this command merely rewinds the tape and resets a few internal program pointers.

READ id N

For the disk, this reads logical block **N** and prints the contents (512 bytes) on the host monitor in hexadecimal format. This command should not be used for the tape drive.

LOCATE id N

For the tape, this is the SCSI LOCATE command, which positions the tape at block **N**. This command should not be used for the disk.

SPACE id N code

SCSI SPACE command. Skip over **N** logical blocks if **code** = 0, or skip over **N** filemarks of **code** = 1, or skip to the end of data if **code** = 3. This is for the tape drive only.

WFILMARK id N

SCSI WRITE FILE MARK command. Write **N** file marks. This is for the tape drive only.

READPOS id

SCSI READ POSITION command for device **id**. Device response data is displayed on the host monitor with data values in hexadecimal.

SETBL id bl

Sets the logical block length equal to **bl** for the tape drive. This should not be used for the disk drive. This command may be necessary before reading a tape.

TWRIT id

Writes 9 files to device **id**, for testing and diagnostic purposes.

TREAD id

Reads the 9 files written by TWRIT, for testing and diagnostic purposes.

DIR [filename]

This command displays the disk directory. The **filename** parameter is optional. If **filename** is absent, the full directory is displayed. If **filename** is present, only those directory entries with names that match **filename** are displayed. **filename** may contain any of three "wild card" characters: "?", "*", and "&". The "?" in **filename** is construed as matching any single character in a directory file name. The "*" is construed as matching any substring without delimiters in the directory file name. A delimiter is any of ":" (colon), "\" (backslash), or "." (period). When the "*" appears in **filename**, characters in the directory file name are skipped until (1) a delimiter is encountered; (2) the end of the directory file name is encountered; or (3)

the character following the "*" in **filename** is encountered in the directory file name. The "&" is construed as matching any substring, with or without delimiters. When the "&" appears in **filename**, characters in the directory file name are skipped until either the end of the directory file name is encountered or the character following the "&" in **filename** is encountered in the directory file name.

The SCSI disk has only one directory; it does not use sub-directories. File names are case sensitive. File names may include any character except space, carriage return (enter), and the nul character, although problems may develop if the wild card characters "&", "*", or "?" are used as part of a file name. The characters ":", "\", and "." can be used in file names. These characters do not have the same special meanings that they have in the DOS file system, but their special interpretation by the wild card character "*" makes some of the file operations look the same as DOS operations. File names may have up to 52 characters, in addition to the device name "SCSIDISK:". This device name is used in programs E and F, but it is not entered as part of the file name in the disk directory. This device name "SCSIDISK:" is normally not used in program TSCSI.

DEL filename

This deletes from the SCSI disk all files with names matching **filename**. This command uses the same wild card conventions that are used in the DIR command. Thus, the command "DEL &" will delete all files from the SCSI disk. The file named "AB:\CD\EFG.xyz" would be deleted by "DEL &xyz" or "DEL AB:&" or "DEL AB:\CD\E*.xyz" or "DEL *:\CD*.*" or "DEL &CD&" but not by "DEL &uvw" or "DEL MN&" or "DEL &PQ&" or "DEL *xyz" or "DEL AB:*" or "DEL *CD*".

COPT2D filename

Copy one file from the SCSI tape drive to SCSI disk file **filename**. The program will ask for the number of records and the record length. The tape should be positioned at the beginning of the source file before this command is issued.

COPD2H SCSIdiskfile hostfile

Copy file **SCSIDiskfile** from the SCSI disk to the host file **hostfile**.

COPH2D hostfile SCSIdiskfile

Copy the host file **hostfile** to the SCSI disk file **SCSIDiskfile**.

REN oldname newname

Change the name of the existing SCSI disk file **oldname** to **newname**. Wild card constructions are permitted; any wild character in **oldname** must be matched by the same character in **newname**.

STOP

Stop execution of TSCSI.

APPENDIX A

EXAMPLE 1: Median Filter Testing

EXAMPLE 1: Median Filter Testing

This sample application is presented primarily for the benefit of the prospective ATR system user, to illustrate the use of the system. In this example, we use only one training scene, and we have only one image per scene. The features used are based on one-dimensional median filtering. More specifically, the difference between the result of median filtering with a window of N pixels and with a window of $N+2$ pixels is used as the meaningful feature. The goal in this application is to find medium-sized lettering and reject large lettering, small lettering, and other kinds of texture.

The training scene image is in file \IMG\DEMO4.IMG on the PC system disk. This image, color coded to indicate the target and background regions specified by the user with program G, is shown in Figure A1. The region selection information is in the mask file \IMG\M4D.IMG, created by program G. The regions designated as targets for the training program include only the medium-sized lettering, in the top left and the bottom right corners of the scene. The regions designated as background include the larger lettering, in the top right corner and at the left edge; some of the small lettering, at the bottom of the scene; and, a large amount of the picture engraving. Note that it is not necessary to mark all of the small lettering, for example, as background; if some of it is marked as background and none of it is marked as target, that should be sufficient. Similarly, it is not necessary to mark every target that may appear in the training scenes. All that is required is that a representative set of examples of targets and of background be marked in the training scenes.

The F command file (.cmd) is fmedflh.cmd, which contains:

```
1 mode value is not used
fmedfl.sl
fmedflh.fc
fmedflh.sum
fmedflh.out
0.5
```

The scene list file (.sl) is fmedfl.sl:

```
1 = number of scenes
1 = number of images per scene
\img\m4d.img
\img\demo4.img
```

The feature calculation file (.fc) is fmedflh.fc:

```
REM USE TYPE 1 MEDIAN FILTERS. fmedflh.fc
REM Detect positive & negative peaks separately.
REM Use square of peak amplitude also.
REM Use a median filter in the perpendicular direction
REM to reduce noise.

defimg 1 512 480 0 ; define image 1, 512x480, no overlap rows.
```

```

copdef 3 1 6 ; define image 3 like image 1 but with 6 overlap rows.
copdef 4 1 6 ; images 3, 4, and 5 are distributed the same as image
copdef 5 1 6 ; 1 and are compatible for multi-image operations.

```

```

REM Generate the MASK feature image, feature image 0,
REM from the scene mask image:
readscene 1 ; read from the file listed first in the .sl file.
undersample 2 1 0 2.0 2.0 0 ; decimate by 2 in each direction.
writefeat 2 ; write MASK feature image (number 0) to disk file
SCSIDISK:\FI\MASK.FI

```

```

REM Read the single scene image:
readscene 1 ; read from next file listed in .sl file.

```

```

REM run length measurements in y (vertical) direction:
copy 3 1 ; copy image 1 to image 3.
REM for run length = 1:
overlap 3 ; share overlap values among nodes.
medly 4 3 3 ; image 4 = image 3 median filtered, 1-D, 3 pixels.
sub 5 3 4 ; image 5 = raw - median filtered.
medlx 3 5 3 ; eliminate 1-pixel wide noise.
REM negative peaks with y-direction run length = 1:
mulcon 3 3 -1.0 ; multiply by -1.
maxcon 5 3 0.0 ; remove negative values.
smthx 5 5 2.0 4 ; smooth in x direction.
smthy 5 5 4.0 4 ; smooth in y direction.
resample 2 5 0 0 0 0 ; decimate, to already-defined image 2.
scale 2 2 ; scale in preparation for writing.
writefeat 2 ; write next feature image (number 1) to disk
SCSIDISK:\FI\Y1N.FI

```

```

REM for run length = 2:
overlap 4 ; do more filtering on once-filtered image:
medly 3 4 5 ; 5-pixel window median filter.
sub 5 4 3 ; difference, 3-pix - 5-pix filtered.
medlx 4 5 3 ; noise reduction.
REM negative peaks:
mulcon 4 4 -1.0
maxcon 5 4 0.0
smthx 5 5 2.0 4
smthy 5 5 4.0 4
resample 2 5 0 0 0 0
scale 2 2
writefeat 2

```

```

SCSIDISK:\FI\Y2N.FI
REM for run length = 3:
overlap 3
medly 4 3 7 ; 7-pixel window.
sub 5 3 4
medlx 3 5 3
mulcon 3 3 -1.0
maxcon 5 3 0.0

```

```

smthx 5 5 2.0 4
smthy 5 5 4.0 4
resample 2 5 0 0 0 0
scale 2 2
writefeat 2
  SCSIDISK:\FI\Y3N.FI

```

REM for run length in x (i, horizontal) direction:

```

copy 3 1
medlx 4 3 3
sub 5 3 4
overlap 5
medly 3 5 3
mulcon 3 3 -1.0
maxcon 5 3 0.0
smthx 5 5 2.0 4
smthy 5 5 4.0 4
resample 2 5 0 0 0 0
scale 2 2
writefeat 2
  SCSIDISK:\FI\X1N.FI
medlx 3 4 5
sub 5 4 3
overlap 5
medly 4 5 3
mulcon 4 4 -1.0
maxcon 5 4 0.0
smthx 5 5 2.0 4
smthy 5 5 4.0 4
resample 2 5 0 0 0 0
scale 2 2
writefeat 2
  SCSIDISK:\FI\X2N.FI
medlx 4 3 7
sub 5 3 4
overlap 5
medly 3 5 3
mulcon 3 3 -1.0
maxcon 5 3 0.0
smthx 5 5 2.0 4
smthy 5 5 4.0 4
resample 2 5 0 0 0 0
scale 2 2
writefeat 2 ; write feature image number 6
  SCSIDISK:\FI\X3N.FI

```

REM new feature images = products of old ones:

```

feat* 1 4 ; product of features 1 and 4.
feat* 2 5
feat* 3 6
feat* 1 5

```

```

feat* 2 6
feat* 2 4
feat* 3 5

```

STOP

This feature calculation file contains instructions that first create a mask which is the same size as the feature images and store the mask as if it were the first (number 0) feature image. The later instructions define 13 feature images, 6 of which are stored on the SCSI disk as image files and 7 of which are defined as products of the others and are not actually stored as files.

The training program F uses these input files and generates files fmedf1h.sum and fmedf1h.out, as specified in the command file. The result output file fmedf1h.out includes optimal coefficient values for each of several sets of included features, and an uncertainty estimate (estimated total error rate) for each set. The graph in Figure A2 summarizes these results. The result output file fmedf1h.out is long, and is listed in part here:

```

for 14 active features:          0.175471
0.175471 = 0.500 * 0.056746 + 0.500 * 0.294196
96.413246 = Q = target criterion.

```

```

f      C[f]      import.
0  6.905470e+01  0.07501  SCSIIDISK:\FI\MASK.FI
1 -1.230949e+01 -0.00922  SCSIIDISK:\FI\Y1N.FI
2  6.688059e+00 -0.01022  SCSIIDISK:\FI\Y2N.FI
3 -6.544309e+00 -0.00922  SCSIIDISK:\FI\Y3N.FI
4  2.173096e+01 -0.00570  SCSIIDISK:\FI\X1N.FI
5  9.023728e+00  0.02863  SCSIIDISK:\FI\X2N.FI
6 -1.139377e+01  0.06507  SCSIIDISK:\FI\X3N.FI
7  9.544310e+00 -0.00920  !*001004
8  1.365871e+00 -0.00704  !*002005
9 -2.541479e+00 -0.00860  !*003006
10 2.975932e+00 -0.00922  !*001005
11 2.677683e+00 -0.00935  !*002006
12 1.240894e+01 -0.00472  !*002004
13 7.573971e+00 -0.00738  !*003005

```

ABNORMAL ERROR DECREASE:

```

for 13 active features:          0.165250
0.165250 = 0.500 * 0.032445 + 0.500 * 0.298054
91.966316 = Q = target criterion.

```

```

f      C[f]
0  6.297519e+01  SCSIIDISK:\FI\MASK.FI
1 -3.901356e+00  SCSIIDISK:\FI\Y1N.FI
3 -2.926445e+00  SCSIIDISK:\FI\Y3N.FI
4  2.314094e+01  SCSIIDISK:\FI\X1N.FI
5  6.831545e+00  SCSIIDISK:\FI\X2N.FI
6 -8.239699e+00  SCSIIDISK:\FI\X3N.FI

```

```

7 -1.702130e+00 !*001004
8 5.421373e+00 !*002005
9 -4.044115e+00 !*003006
10 -1.274309e+00 !*001005
11 7.129105e-01 !*002006
12 2.498617e+01 !*002004
13 7.000552e+00 !*003005

```

RESTART:

```

for 14 active features:      0.164748
0.164748 = 0.500 * 0.027901 + 0.500 * 0.301596
90.807426 = Q = target criterion.

```

```

f      C[f]      import.
0 6.165407e+01 0.08525 SCSIDISK:\FI\MASK.FI
1 -1.997215e+00 0.00045 SCSIDISK:\FI\Y1N.FI
2 -1.405803e+00 0.00040 SCSIDISK:\FI\Y2N.FI
3 -2.365844e+00 0.00035 SCSIDISK:\FI\Y3N.FI
4 2.329933e+01 0.02221 SCSIDISK:\FI\X1N.FI
5 5.318080e+00 0.01103 SCSIDISK:\FI\X2N.FI
6 -6.925702e+00 0.04903 SCSIDISK:\FI\X3N.FI
7 -5.237159e+00 0.00026 !*001004
8 7.290884e+00 0.02268 !*002005
9 -3.500515e+00 0.00060 !*003006
10 -1.582983e+00 -0.00007 !*001005
11 -9.228410e-01 -0.00001 !*002006
12 2.825508e+01 0.04749 !*002004
13 6.175331e+00 0.00115 !*003005

```

FILE TEXT IS MISSING HERE

```

for 6 active features:      0.170221
0.170221 = 0.500 * 0.033679 + 0.500 * 0.306763
93.000191 = Q = target criterion.

```

```

f      C[f]      import.
0 6.063261e+01 0.07979 SCSIDISK:\FI\MASK.FI
1 -3.885988e+00 0.00298 SCSIDISK:\FI\Y1N.FI
4 4.440161e+01 0.06733 SCSIDISK:\FI\X1N.FI
6 -7.864605e+00 0.05273 SCSIDISK:\FI\X3N.FI
8 1.664656e+01 0.08818 !*002005
9 -2.005543e+00 0.00027 !*003006

```

```

for 5 active features:      0.170490
0.170490 = 0.500 * 0.034567 + 0.500 * 0.306414
93.142075 = Q = target criterion.

```



```

f      C[f]      import.
0  6.006615e+01  0.07953 SCSIDISK:\FI\MASK.FI
1 -3.642586e+00  0.00286 SCSIDISK:\FI\Y1N.FI
4  4.433332e+01  0.06753 SCSIDISK:\FI\X1N.FI
6 -8.006670e+00  0.05427 SCSIDISK:\FI\X3N.FI
8  1.626991e+01  0.05714 !*002005

```

```

for 4 active features:      0.173354
0.173354 = 0.500 * 0.035825 + 0.500 * 0.310883
93.122307 = Q = target criterion.

```

```

f      C[f]      import.
0  5.473910e+01  0.07667 SCSIDISK:\FI\MASK.FI
4  4.621682e+01  0.06988 SCSIDISK:\FI\X1N.FI
6 -7.831959e+00  0.05222 SCSIDISK:\FI\X3N.FI
8  1.643995e+01  0.05579 !*002005

```

```

for 3 active features:      0.225576
0.225576 = 0.500 * 0.159022 + 0.500 * 0.292130
86.924637 = Q = target criterion.

```

```

f      C[f]      import.
0  4.786732e+01  0.02599 SCSIDISK:\FI\MASK.FI
4  4.702850e+01 -0.02771 SCSIDISK:\FI\X1N.FI
8  1.672868e+01  0.08130 !*002005

```

```

for 2 active features:      0.197867
0.197867 = 0.500 * 0.033910 + 0.500 * 0.361824
98.398445 = Q = target criterion.

```

```

f      C[f]      import.
0  5.563923e+01  9.99990 SCSIDISK:\FI\MASK.FI
8  2.536514e+01  9.99990 !*002005

```

there are no acceptable smaller feature subsets.

At this point, a new coefficient calculation is done for a different value of the type 1 error weight, 0.90. The command file (.cmd) is fmedf1h2.cmd:

```

1 mode value is not used
fmedf1h.sum
NUL
NUL
fmedf1h2.out
0.9

```

This command file gives the name of the sum file output by the previous F calculation, in

place of a scene list file. The result of this is that this F calculation does not access any image files and does not calculate any feature images, thereby saving a lot of time. This F calculation merely uses the sums from a previous calculation and does a new optimization for the new value of the type 1 error weight. This F calculation outputs its results to file fmedf1h2.out, which is listed in part here:

```
for 14 active features:          0.039632
0.039632 = 0.900 * 0.002812 + 0.100 * 0.371013
106.356369 = Q = target criterion.
```

```
f      C[f]      import.
0  6.240343e+01  0.01038 SCSIDISK:\FI\MASK.FI
1 -4.529906e+00 -0.00146 SCSIDISK:\FI\Y1N.FI
2  2.635504e+00 -0.00144 SCSIDISK:\FI\Y2N.FI
3 -4.813251e+00 -0.00154 SCSIDISK:\FI\Y3N.FI
4  2.167990e+01  0.00541 SCSIDISK:\FI\X1N.FI
5  6.383419e+00  0.00403 SCSIDISK:\FI\X2N.FI
6 -7.766041e+00 -0.00092 SCSIDISK:\FI\X3N.FI
7  2.323600e+00 -0.00143 !*001004
8  3.679545e+00 -0.00058 !*002005
9 -2.296764e+00 -0.00144 !*003006
10 -9.088939e-01 -0.00158 !*001005
11 -1.956589e+00 -0.00133 !*002006
12  2.865775e+01  0.01126 !*002004
13  7.252297e+00  0.00331 !*003005
```

```
.
.  FILE TEXT IS MISSING HERE
.
```

```
for 6 active features:          0.038530
0.038530 = 0.900 * 0.001673 + 0.100 * 0.370237
100.068092 = Q = target criterion.
```

```
f      C[f]      import.
0  6.154808e+01  0.01147 SCSIDISK:\FI\MASK.FI
2 -5.170771e+00  0.00154 SCSIDISK:\FI\Y2N.FI
6 -3.419756e-01  0.00004 SCSIDISK:\FI\X3N.FI
8  1.368567e+01  0.01216 !*002005
11 -8.860680e+00  0.00167 !*002006
12  4.439417e+01  0.01622 !*002004
```

```
for 5 active features:          0.038571
0.038571 = 0.900 * 0.001691 + 0.100 * 0.370489
100.089783 = Q = target criterion.
```

```
f      C[f]      import.
0  6.126400e+01  0.01143 SCSIDISK:\FI\MASK.FI
2 -5.171503e+00  0.00150 SCSIDISK:\FI\Y2N.FI
8  1.372271e+01  0.01019 !*002005
11 -8.929868e+00  0.00195 !*002006
12  4.443060e+01  0.01425 !*002004
```

```

for 4 active features:      0.040074
0.040074 = 0.900 * 0.001952 + 0.100 * 0.383173
103.289093 = Q = target criterion.

```

```

f      C[f]      import.
0  5.779496e+01  0.00993 SCSIDISK:\FI\MASK.FI
8  1.293844e+01  0.01008 !*002005
11 -9.245314e+00  0.00216 !*002006
12  4.276791e+01  0.01099 !*002004

```

```

for 3 active features:      0.042236
0.042236 = 0.900 * 0.002378 + 0.100 * 0.400960
108.795937 = Q = target criterion.

```

```

f      C[f]      import.
0  5.654004e+01  0.00779 SCSIDISK:\FI\MASK.FI
8  1.004776e+01  0.00780 !*002005
12  3.924446e+01  0.00362 !*002004

```

```

for 2 active features:      0.045858
0.045858 = 0.900 * 0.003053 + 0.100 * 0.431104
115.866432 = Q = target criterion.

```

```

f      C[f]      import.
0  5.563923e+01  9.99990 SCSIDISK:\FI\MASK.FI
8  2.536514e+01  9.99990 !*002005

```

there are no acceptable smaller feature subsets.

Figure A2: Total (weighted sum) error rate versus number of features for two program F optimization calculations.

Figure A2 shows the error rate versus number of features for both of the two F runs. The error rate values above 0.15 are for the first run, and the values below 0.05 are for the second run.

In this example, the coefficients chosen for use in the surveillance process are those for 5 features in the second F calculation. The E command file (.cmd) is emedf1h.cmd:

```
1 = mode. this parameter is not used in E.
emedf1h.sl
emedf1h.fc
NUL
```

This E command file specifies "NUL" for the output file. The current version of E does not use an output file, but an output file name should still be specified.

The E scene list file (.sl) is emedf1h.sl:

```
4 = number of scenes
1 = number of images per scene
\ri\demol.ri
```

\img\demo1.img

+
+
+
+
+
+

This scene list file gives "demo1" as the base file name for both the output result image file and the input scene image file for the first scene. For the next 3 scenes, these file names are incremented, so that the base file names for the next 3 scenes are "demo2", "demo3", and "demo4".

The E feature calculation file (.fc) is emedflh.fc:

```
REM USE TYPE 1 MEDIAN FILTERS. fmedflh.fc
REM Detect positive & negative peaks separately.
REM Use square of peak amplitude also.
REM Use a median filter in the perpendicular direction
REM to reduce noise.
```

```
defimg 1 512 480 0 ; define image 1, 512x480, no overlap rows.
copdef 3 1 6 ; define image 3 like image 1 but with 6 overlap rows.
copdef 4 1 6 ; images 3, 4, and 5 are distributed the same as image
copdef 5 1 6 ; 1 and are compatible for multi-image operations.
undersample 7 1 0 2.0 2.0 0 ; half-size result image.
copdef 6 7 0
copdef 2 7 0
```

```
zeroimage 7 ; initiallize result image.
```

```
REM 0 6.126400e+01 0.01143 SCSIDISK:\FI\MASK.FI
addcon 7 7 61.264 ; feature 0, additive constant.
```

```
REM Read the single scene image:
readscene 1 ; read from next file listed in .sl file.
resample 2 1 0 0 0 0 ; image 2 = decimated scene image 1.
ttg3 2 50 128 0.0 ; display decimated scene image; why not?
```

```
REM run length measurements in y (vertical) direction:
copy 3 1 ; copy image 1 to image 3.
REM for run length = 1:
overlap 3 ; share overlap values among nodes.
medly 4 3 3 ; image 4 = image 3 median filtered, 1-D, 3 pixels.
REM for run length = 2:
overlap 4 ; do more filtering on once-filtered image:
medly 3 4 5 ; 5-pixel window median filter.
sub 5 4 3 ; difference, 3-pix - 5-pix filtered.
medlx 4 5 3 ; noise reduction.
REM negative peaks:
```

```

mulcon 4 4 -1.0
maxcon 5 4 0.0
smthx 5 5 2.0 4
smthy 5 5 4.0 4
resample 6 5 0 0 0 0 ; image 6 = feature 2.
REM 2 -5.171503e+00 0.00150 SCSIIDISK:\FI\Y2N.FI
mulcon 2 6 -5.171503 ; image 2 = feature 2 * coefficient.
add 7 7 2 ; accumulate to result image, image 7.

```

```

REM for run length in x (i, horizontal) direction:
copy 3 1
medlx 4 3 3
sub 5 3 4
overlap 5
medly 3 5 3
mulcon 3 3 -1.0
maxcon 5 3 0.0
smthx 5 5 2.0 4
smthy 5 5 4.0 4
resample 2 5 0 0 0 0 ; image 2 = feature 4.
mul 2 2 6 ; image 2 = feature 4 * feature 2 = feature 12.
REM 12 4.443060e+01 0.01425 !*002004
mulcon 2 2 44.4306 ; image 2 = feature 12 * coefficient.
add 7 7 2

```

```

medlx 3 4 5
sub 5 4 3
overlap 5
medly 4 5 3
mulcon 4 4 -1.0
maxcon 5 4 0.0
smthx 5 5 2.0 4
smthy 5 5 4.0 4
resample 2 5 0 0 0 0 ; feature 5.
mul 2 2 6 ; feature 8 = feature 5 * feature 2.
REM 8 1.372271e+01 0.01019 !*002005
mulcon 2 2 13.72271 ; feature 8 * coefficient.
add 7 7 2

```

```

medlx 4 3 7
sub 5 3 4
overlap 5
medly 3 5 3
mulcon 3 3 -1.0
maxcon 5 3 0.0
smthx 5 5 2.0 4
smthy 5 5 4.0 4
resample 2 5 0 0 0 0 ; feature 6.
mul 2 2 6 ; feature 11 = feature 6 * feature 2.
REM 11 -8.929868e+00 0.00195 !*002006
mulcon 2 2 -8.929868 ; feature 11 * coefficient.

```

```
add 7 7 2
```

```
REM Eliminate false positives from median filter edge effects:
```

```
defkern 1 -3 3 -3 3
```

```
zeroimage 2
```

```
copyedges 7 2 1 ; copy zeroed edges from image 2 to image 7.
```

```
maxcon 7 7 0.0 ; remove values < 0.
```

```
mincon 7 7 255.0 ; remove values > 255.
```

```
REM 100.089783 = Q = target criterion.
```

```
writeresult 7 1 100.089783 ; write result image.
```

```
ttg3 7 50 768 100.089783 ; display result image.
```

```
STOP
```

This E feature calculation file is used for each of the four scenes specified in the scene list file. It includes the values of the coefficients from the F result output file for 5 features.

This E calculation generates four result images. These result images are smaller than the scene images, having half as many pixels in each direction. This of course implies half the spatial resolution in each direction, but result images usually do not require much spatial resolution so time and disk space can be saved by using the lower resolution for result images. The four result images are shown in Figures A3 - A6, along with their corresponding scene images decimated to the same size as the result images. (The combination of decimation and reproduction for this report greatly degrades the quality of these scene images.) In the result images, the bright, green to pale green to white, regions indicate targets; the dark, red to black, regions indicate background. The lighter regions are stronger indications of target. Figure A6 shows that the ATR system did correctly designate the medium-sized lettering as target, and it did correctly reject the larger letters as background. The results are mixed for the small letters at the bottom center of the scene, some being correctly rejected as background and some being incorrectly identified as targets. A few other regions in the DEMO4 scene, as well as a few small regions in the other scenes, are incorrectly identified as targets by this ATR system. These incorrect identifications simply mean that this ATR system, with the particular features which were chosen for this application, cannot distinguish between medium-sized lettering and whatever is shown in the scene images at those regions incorrectly identified as targets. Presumably, the operator could choose a better set of features and reduce the incidence of errors in this ATR application. Note, however, that this relatively small set of relatively simple features did quite well: there are essentially no type 1 errors (incorrect designation of targets as background), and there are not many type 2 errors (incorrect designation of background as target). The absence of type 1 errors is largely due to the large value (0.9) chosen for the type 1 error weight in the F optimization calculation.

APPENDIX B

EXAMPLE 2: Roads and Riverbanks

EXAMPLE 2: Roads and Riverbanks

This simple example illustrates fusion of image data, as well as some aspects of using the software. We have two images of the same scene, one visible light and one infrared image. (These images are part of a set supplied by Karen Steinmaus of Battelle, Pacific Northwest Laboratories, one of the participants in the Department of Energy's Airborne Multisensor Pod System project.) The visible light image (Figure B1) shows roads quite clearly, but it also shows riverbanks and it is difficult to distinguish between the two features in this image. The infrared image (Figure B2) does not show the roads very well, but it clearly indicates where the river is. The two images together should allow us to find roads and reject riverbanks. Note, however, that this cannot be done by looking for roads in each of the two images separately and then simply adding or averaging the two results; a more sophisticated approach to image data fusion, such as that used in this ATR system, is required.

For this illustration, we do a very simple analysis using only 4 features. For the first feature, we do a convolution of the visible light image with a 13x13 pixel kernel in which the pixel values are proportional to X^2 with the mean subtracted out, clip the result to keep only negative values, and take the absolute value. This convolution kernel is in file c:\imanal\opr\pnlopr\pnlxx.opr:

```
22. 11. 2. -5. -10. -13. -14. -13. -10. -5. 2. 11. 22.
22. 11. 2. -5. -10. -13. -14. -13. -10. -5. 2. 11. 22.
22. 11. 2. -5. -10. -13. -14. -13. -10. -5. 2. 11. 22.
22. 11. 2. -5. -10. -13. -14. -13. -10. -5. 2. 11. 22.
22. 11. 2. -5. -10. -13. -14. -13. -10. -5. 2. 11. 22.
22. 11. 2. -5. -10. -13. -14. -13. -10. -5. 2. 11. 22.
22. 11. 2. -5. -10. -13. -14. -13. -10. -5. 2. 11. 22.
22. 11. 2. -5. -10. -13. -14. -13. -10. -5. 2. 11. 22.
22. 11. 2. -5. -10. -13. -14. -13. -10. -5. 2. 11. 22.
22. 11. 2. -5. -10. -13. -14. -13. -10. -5. 2. 11. 22.
22. 11. 2. -5. -10. -13. -14. -13. -10. -5. 2. 11. 22.
22. 11. 2. -5. -10. -13. -14. -13. -10. -5. 2. 11. 22.
```

This first feature indicates the presence of both vertical roads and vertical riverbanks.

For the third feature, we do a convolution of the infrared image with a 13x13 pixel kernel in which the pixel values are proportional to X , and square the result. This kernel is in file c:\imanal\opr\pnlopr\pnlx.opr:

```
-6. -5. -4. -3. -2. -1. 0. 1. 2. 3. 4. 5. 6.
-6. -5. -4. -3. -2. -1. 0. 1. 2. 3. 4. 5. 6.
-6. -5. -4. -3. -2. -1. 0. 1. 2. 3. 4. 5. 6.
-6. -5. -4. -3. -2. -1. 0. 1. 2. 3. 4. 5. 6.
-6. -5. -4. -3. -2. -1. 0. 1. 2. 3. 4. 5. 6.
-6. -5. -4. -3. -2. -1. 0. 1. 2. 3. 4. 5. 6.
-6. -5. -4. -3. -2. -1. 0. 1. 2. 3. 4. 5. 6.
```

```

-6. -5. -4. -3. -2. -1. 0. 1. 2. 3. 4. 5. 6.
-6. -5. -4. -3. -2. -1. 0. 1. 2. 3. 4. 5. 6.
-6. -5. -4. -3. -2. -1. 0. 1. 2. 3. 4. 5. 6.
-6. -5. -4. -3. -2. -1. 0. 1. 2. 3. 4. 5. 6.
-6. -5. -4. -3. -2. -1. 0. 1. 2. 3. 4. 5. 6.

```

This third feature is insensitive to roads, but shows vertical riverbanks very well.

The second and fourth features are analagous to the first and third, using kernels with dependence on the Y coordinate instead of the X coordinate to detect roads and riverbanks that run horizontally in the images instead of vertically.

Figures B1 and B2 both show the operator-selected target (green) and background (red) regions. Note that these regions do not need to be marked on both of the scene images separately; it is sufficient to use either one of the scene images with the program G to mark the target and background regions.

The F program command file is

```

0 unused mode code
fpnl1.sl
fpnl2.fc
fpnl2.sum
fpnl2.out
0.5

```

and the scene list file (fpnl1.sl) is

```

1 scene
2 images per scene
scsidisk:PNLmsk03.img
scsidisk:PNLls01.img
scsidisk:PNL6s01.img

```

and the feature calculation file (fpnl2.fc) is

```

echo PNL01 test
defimg 1 512 512 6; image 1 has 512 rows, 512 columns, 6 overlap
defkern 1 -6 6 -6 6; operator (kernel) 1 is 13x13 pixels.
copdef 3 1 0; image 3 is the same size as image 1, zero overlap.
zeroimage 3

featfil ; use scene mask as feature mask.

readscene 1; image 1 is first scene image.
readkernel 1; read convolution operator (kernel).
c:\imanal\opr\pnlopr\PNLxx.opr
convolve 3 1 1; image 3 = image 1 convolved with kernel 1.
mincon 3 3 0.0; keep only negative results, set positive to zero.
abs 3 3; absolute value.
scale 3 3 1; prepare image 3 for output as a feature image.
writefeat 3; output to disk, put file name in internal list.

```

```

D:tem1.img

readkernel 1; new kernel, same old scene image.
  c:\imanal\opr\pnlopr\PNLy.y.opr
convolve 3 1 1
mincon 3 3 0.0
abs 3 3
scale 3 3 1
writefeat 3; automatically select feature image file name:
+

readscene 1; new scene image.
readkernel 1; new kernel.
  c:\imanal\opr\pnlopr\PNLx.opr
convolve 3 1 1
mul 3 3 3; multiply result by itself.
scale 3 3 1
writefeat 3
+

readkernel 1
  c:\imanal\opr\pnlopr\PNLy.y.opr
convolve 3 1 1
mul 3 3 3
scale 3 3 1
writefeat 3
+

STOP

```

The part of the F program output file that is used for the E program is

```

for 5 active features:      0.168050
0.168050 = 0.500 * 0.023600 + 0.500 * 0.312500
90.007324 = Q = target criterion.

f      C[f]      import.
0  5.957882e+01  0.08195 scsidisk:PNLmsk03.img
1  2.562973e-03  0.02468 D:tem1.img
2  1.270594e-02  0.12367 D:tem2.img
3  -7.132083e-08  0.01399 D:tem3.img
4  -1.236135e-07  0.00208 D:tem4.img

```

The surveillance program E command file is

```

1 mode value is not used
epnl2.sl
epnl2.fc
NUL

```

and the scene list file (epnl2.sl) is

```
4 four scenes
2 two images per scene
\ri\epnl2a.ri
scsidisk:PNLtapel\imagela
scsidisk:PNLtapel\image6a
+
+1
+2
+
+1
+2
+
+1
+2
```

This scene list file specifies 4 scenes, with the names of corresponding files for each scene being the previous file names incremented.

The E feature calculation file (epnl2.fc) is

echo PNL01 test

```
udefF32 Q COEF0 COEF1 COEF2 COEF3 COEF4; feature coefficients:
REM The following 6 lines are copied from the F output file
REM and modified to define coefficient values:
ueq Q 90.007324 = Q = target criterion.
ueq COEF0 5.957882e+01 0.08195 scsidisk:PNLmsk03.img
ueq COEF1 2.562973e-03 0.02468 D:tem1.img
ueq COEF2 1.270594e-02 0.12367 D:tem2.img
ueq COEF3 -7.132083e-08 0.01399 D:tem3.img
ueq COEF4 -1.236135e-07 0.00208 D:tem4.img
```

udefI32 PNLxx PNLyy PNLx PNLy; kernel numbers:

```
ueq PNLxx 1
ueq PNLyy 2
ueq PNLx 3
ueq PNLy 4
defkern PNLxx -6 6 -6 6
defkern PNLyy -6 6 -6 6
defkern PNLx -6 6 -6 6
defkern PNLy -6 6 -6 6
readkernel PNLxx
c:\imanal\opr\pnlopr\PNLxx.opr
readkernel PNLyy
c:\imanal\opr\pnlopr\PNLyy.opr
readkernel PNLx
c:\imanal\opr\pnlopr\PNLx.opr
readkernel PNLy
c:\imanal\opr\pnlopr\PNLy.opr
```

```

.
undefI32 SRC1 SRC2 RESULT SKR1; image numbers:
ueq SRC1 1; first scene image.
ueq SRC2 2; second scene image.
ueq RESULT 3; result image.
ueq SKR1 4; scratch image.
defimg SRC1 512 512 6
copdef SRC2 SRC1 6
copdef RESULT SRC1 0
copdef SKR1 SRC1 0

readscene SRC1
readscene SRC2
zeroimage RESULT
zeroimage SKR1

convolve SKR1 SRC1 PNLxx; convolve scene image with kernel.
mincon SKR1 SKR1 0.0
abs SKR1 SKR1
mulcon SKR1 SKR1 COEF1; feature 1.
add RESULT RESULT SKR1; accumulate feature*coefficient to result.

convolve SKR1 SRC2 PNLx
mul SKR1 SKR1 SKR1
mulcon SKR1 SKR1 COEF3; feature 3.
add RESULT RESULT SKR1

convolve SKR1 SRC1 PNLyy
mincon SKR1 SKR1 0.0
abs SKR1 SKR1
mulcon SKR1 SKR1 COEF2; feature 2.
add RESULT RESULT SKR1

convolve SKR1 SRC2 PNLy
mul SKR1 SKR1 SKR1
mulcon SKR1 SKR1 COEF4; feature 4.
add RESULT RESULT SKR1

addcon RESULT RESULT COEF0; additive constant, feature 0.

maxcon RESULT RESULT 0.0; limit result pixel values, > 0
mincon RESULT RESULT 254.0; and < 255.
zeroimage SKR1; zero edge rows and columns in result;
copyedges RESULT SKR1 PNLx; they don't mean anything.
writeresult RESULT 1 Q; write with header type 1.
TTG3 RESULT 0 0 Q; display result, label = filename:
.

STOP

```

This feature calculation file uses some user-defined variables to make the file more readable and to make it easier to change the values of the coefficients from the F program.

The result image for one of the scenes, the same scene which was used as the training scene, is shown in Figure B3. In this result image, the white is the strongest indication of roads, darker green is a weaker indication of roads, black is the strongest indication of background (non-road), and lighter red is a weaker indication of background. This result does distinguish clearly between road and riverbank, thus satisfying the goal for this illustration: the two images analyzed jointly give a clear indication of a result that is not obvious in either image separately. This result indicates horizontal roads more strongly than vertical roads; this is because the horizontal roads appear narrower in these images, and the convolution calculations used here are more sensitive to narrower features. This result also indicates as roads some regions that are neither road nor riverbank; this is not surprising, since no significant effort was made to exclude miscellaneous clutter from being identified as target in this very simple example.

APPENDIX C

EXAMPLE 3: Buried Waste Location

EXAMPLE 3: Buried Waste Location

This example illustrates a more intensive use of user-defined scalar variables; some image manipulation methods; and, the use of a completely naive but still quite effective analysis method in which the ATR system's capabilities replace operator understanding of the data. The primary purpose of this study was to assess the utility of this ATR system in buried waste recovery operations.

Data

In this application, as many as 7 sensors acquired different types of data. This data is not image data in the traditional sense. Rather, each "image" is merely a set of values measured at a two-dimensional array of points on the surface of the ground. Sensor S1A is the vertical component of the earth's magnetic field, and S1B is the gradient (derivative with respect to vertical position) of the vertical component of the magnetic field. S2A, S3A, S4A, and S5A are measurements of the electrical conductivity of the soil, like eddy current measurements, taken with different combinations of field orientation and phase shift. S6A is a volatile chemical sensor. Most scenes include measurements from the first six sensors. Because data from the seventh sensor S6A was available for only 3 of the 35 scenes, and for one of these three scenes no other sensor data was available, this seventh sensor data was not used in this brief study.

There were 5 experiments, referred to as E1, E2, E3, E5, and E6. The different experiments represent data recorded over five different areas with different buried objects. The five rectangles on the left of Figure C1 indicate the *approximate* sizes and locations of buried objects in the five experiments, as seen from above. (North is to the left.) In these sketches, the solid objects are magnetic, and the objects drawn with outlines only are, at least mostly, not magnetic. In E2, for example, there are a magnetic barrel (object 7) and a wooden box (object 6). Objects 9 and 10 in E3 are two boxes, one above the other, with some magnetic material in the top box. The top box was removed between L3 and L4 (to be described later). Similarly, object 20 (a vertical barrel) in E5 was removed between L2 and L3, and object 27 (a vertical steel pipe) in E6 was removed between L2 and L3.

For each experiment, there are several different levels or vertical positions of the sensors. The number of different levels is not the same for all experiments. The several levels are referred to as L0, L1, L0 is the highest level, and the sensors are 6 inches lower for each successive level. In some cases, layers of soil were removed between successive measurement levels. In these analyses, any one level of any one experiment is treated as a separate *scene*. Different scenes are expected to give different results for any measurement, because they contain different objects or because the objects are at different distances from the sensors. There are a total of 34 usable scenes in this data set.

The scene images in this data have 43 columns (43 x values, spaced 3 inches apart) and 25 rows (25 y values, spaced 6 inches apart). Some of the data sets did not have this many rows or columns, and some were missing a few data points from what was expected to be a regularly-spaced array. In all of these cases, the missing data points were filled in using a linear interpolation or extrapolation procedure, NTRP01, so that each scene image used in this study had a full $43 \times 25 = 1075$ data points. The interpolation was done separately and is

not included in the feature calculation files listed here.

Analyses

Perhaps the simplest analysis we can use with this ATR system for a set of data with 6 sensors is a linear combination of the 6 raw data images (augmented with the "constant" $f=0$ image which is always included for mathematical completeness). This simple linear analysis was tried with the buried waste data. The training was done with 4 scenes: E2L7, E5L2, E5L5, and E6L5. The masks used were M2A, M5A, M5B, and M6A, shown in Figure C1. In this figure, the red (actually more brown) areas of the masks are designated as background, the green areas are target, and the black areas are not used in the training process. These masks mark barrels as targets and other regions as background; that is, this analysis is a search for barrels. The results of this training process were used to analyze 30 scenes from experiments E2, E3, E5, and E6; experiment E1 was not included in this analysis because this analysis uses 6 sensors and only 4 sensors were used in experiment E1. The results are indicated in Figure C3. In this figure, as in any of the result images, lighter regions are stronger indications of targets, and darker regions are stronger indications of background. Green and white (that is, very light green) indicate what the ATR system classifies as targets. Red and brown and black (dark red) indicate what the ATR system classifies as background. Figure C3 shows that this analysis did correctly identify the barrel (object 7) as a target in E2 when the sensors were close to the barrel, levels L6-L8. It also incorrectly identified a box (object 6) as a target in these same scenes. It similarly identified barrels (objects 17 and 20) and mis-identified other objects in other scenes. We must conclude that this simple linear analysis is very poor at distinguishing between barrels and other objects.

The next obvious level of complication in analysis is to add quadratic terms to the linear analysis, adding 21 product terms to the linear and constant terms in the previous analysis. This was tried using the same training set described for the linear case, with the result shown in Figure C4. This quadratic analysis is better than the linear analysis at distinguishing barrels from background, but it is still not as good as one would like.

We list here some files associated with this quadratic analysis. The F command file is f4i.cmd:

```
0 this mode code is not used by F8
f4g.sl
f4i.fc
f4i.sum
f4i.out
0.5
look for barrels
```

The F scene list file is f4g.sl:

```
4 scenes
6 images per scene
\img\bwid1\m2a.big
\img\bwid1\e2l7slb.img
```

```

\img\bwid1\e217s1a.img
\img\bwid1\e217s2a.img
\img\bwid1\e217s3a.img
\img\bwid1\e217s4a.img
\img\bwid1\e217s5a.img
\img\bwid1\m5a.big
\img\bwid1\e512s1b.img
\img\bwid1\e512s1a.img
\img\bwid1\e512s2a.img
\img\bwid1\e512s3a.img
\img\bwid1\e512s4a.img
\img\bwid1\e512s5a.img
\img\bwid1\m5b.big
\img\bwid1\e515s1b.img
\img\bwid1\e515s1a.img
\img\bwid1\e515s2a.img
\img\bwid1\e515s3a.img
\img\bwid1\e515s4a.img
\img\bwid1\e515s5a.img
\img\bwid1\m6a.big
\img\bwid1\e615s1b.img
\img\bwid1\e615s1a.img
\img\bwid1\e615s2a.img
\img\bwid1\e615s3a.img
\img\bwid1\e615s4a.img
\img\bwid1\e615s5a.img

```

The F feature calculation file is f4i.fc:

```

; look for barrels.
; use all quadratic combination of measured values, min subtracted.

```

```

;minimum sensor readings:

```

```

udefF32 S1MIN S2MIN S3MIN S4MIN S5MIN S6MIN S7MIN

```

```

    ueq S1MIN -300000. -270801.
    ueq S2MIN      0.0   6472.7
    ueq S3MIN   -300.   -201.8
    ueq S4MIN   -10.    -9.
    ueq S5MIN  -1000.  -807.1
    ueq S6MIN   -80.   -57.
    ueq S7MIN   -30.   -21.9

```

```

udefI32 NCOL NROW NOVL ; scene image size

```

```

    ueq NCOL 43
    ueq NROW 25
    ueq NOVL 2

```

```

udefI32 BIGMASK MASK SI1 SI2 SI3 SI4 SI5 SI6 SKR1 ; image numbers

```

```

    ueq BIGMASK 1
    ueq MASK 2
    ueq SI1 1
    ueq SI2 2

```

```

ueq SI3 3
ueq SI4 4
ueq SI5 5
ueq SI6 6
ueq SKR1 7

; input, resample (resize), and output mask:
readscene BIGMASK
resample MASK BIGMASK 0 NCOL NROW 0
writefeat MASK ; feature image 0 = mask
  d:MASK.img
defimg BIGMASK 0 0 0 ; un-define image
defimg MASK 0 0 0

; input all scene images, reading and using scale factors:
readscene SI1
;since image SI1 is not previously defined, and no header code is
;given in this READSCENE command, the read operation sets the
;values of the user-accessible variables $A and $B. Use these
;values to restore the original numeric data values:
subcon SI1 SI1 $A
divcon SI1 SI1 $B
;subtract minimum value, make all readings positive:
subcon SI1 SI1 S1MIN
readscene SI2
subcon SI2 SI2 $A
divcon SI2 SI2 $B
subcon SI2 SI2 S2MIN
readscene SI3
subcon SI3 SI3 $A
divcon SI3 SI3 $B
subcon SI3 SI3 S3MIN
readscene SI4
subcon SI4 SI4 $A
divcon SI4 SI4 $B
subcon SI4 SI4 S4MIN
readscene SI5
subcon SI5 SI5 $A
divcon SI5 SI5 $B
subcon SI5 SI5 S5MIN
readscene SI6
subcon SI6 SI6 $A
divcon SI6 SI6 $B
subcon SI6 SI6 S6MIN

;define a scratch image compatible with the scene images:
copdef SKR1 SI1 NOVL

; use absolute magnitude as a feature:
scale SKR1 SI1
stats SKR1

```

```

writefeat SKR1 ; feature image 1, - scene image 1
  d:RAW1.img
scale SKR1 SI2
writefeat SKR1 ; feature image 2
  +
scale SKR1 SI3
writefeat SKR1
  +
scale SKR1 SI4
writefeat SKR1
  +
scale SKR1 SI5
writefeat SKR1
  +
scale SKR1 SI6
writefeat SKR1
  +

```

```

;that is all the linear features.
;now do quadratic features, products of linear:

```

```

feat* 1 1
feat* 2 2
feat* 3 3
feat* 4 4
feat* 5 5
feat* 6 6
feat* 1 2
feat* 1 3
feat* 1 4
feat* 1 5
feat* 1 6
feat* 2 3
feat* 2 4
feat* 2 5
feat* 2 6
feat* 3 4
feat* 3 5
feat* 3 6
feat* 4 5
feat* 4 6
feat* 5 6

```

```

stop

```

The E command file is e4i.cmd:

```

0
e4h.sl
e4i.fc
NUL

```

look for barrels

The E scene list file is e4h.sl, for which we list only the first 16 of 212 lines, representing the first 2 of 30 scenes:

```
30 scenes
6 images per scene
\ri\e210.img
\img\bwid1\e210s1b.img
\img\bwid1\e210s1a.img
\img\bwid1\e210s2a.img
\img\bwid1\e210s3a.img
\img\bwid1\e210s4a.img
\img\bwid1\e210s5a.img
\ri\e211.img
\img\bwid1\e211s1b.img
\img\bwid1\e211s1a.img
\img\bwid1\e211s2a.img
\img\bwid1\e211s3a.img
\img\bwid1\e211s4a.img
\img\bwid1\e211s5a.img
.
  FILE TEXT IS MISSING HERE
.
```

The E feature calculation file is e4i.fc:

```
; look for barrels.
; use all linear & quadratic combinations of
; measurement - minimum.
```

echo_off

```
;coefficients from the F program:
undefF32 Q COEF0 COEF1 COEF2 COEF3 COEF4 COEF5 COEF6
undefF32 COEF7 COEF8 COEF9 COEF10 COEF11 COEF12
undefF32 COEF13 COEF14 COEF15 COEF16 COEF17 COEF18
undefF32 COEF19 COEF20 COEF21 COEF22 COEF23 COEF24
undefF32 COEF25 COEF26 COEF27
  ueq Q 96.469131 = Q = target criterion.
  ueq COEF0 1.451913e+03 0.21729 d:MASK.img
  ueq COEF1 4.511122e-04 0.15536 d:RAW1.img
  ueq COEF2 -2.152343e-02 0.20731 d:RAW2.img
  ueq COEF3 -8.356287e-01 0.23020 d:RAW3.img
  ueq COEF4 -6.656775e+00 0.18433 d:RAW4.img
  ueq COEF5 -7.544662e-01 0.40221 d:RAW5.img
  ueq COEF6 -8.950671e+00 0.25411 d:RAW6.img
  ueq COEF7 -2.263084e-09 0.04445 !*001001
  ueq COEF8 -2.028773e-08 0.12799 !*002002
  ueq COEF9 5.670000e-06 0.00007 !*003003
  ueq COEF10 -5.817067e-02 0.22490 !*004004
```

```

ueq COEF11 1.921685e-04 0.08690 !*005005
ueq COEF12 -4.608979e-03 0.16511 !*006006
ueq COEF13 1.234254e-09 0.00172 !*001002
ueq COEF14 -2.278859e-06 0.21065 !*001003
ueq COEF15 3.241597e-05 0.24932 !*001004
ueq COEF16 6.679167e-07 0.13909 !*001005
ueq COEF17 7.707375e-06 0.20176 !*001006
ueq COEF18 1.938664e-05 0.23094 !*002003
ueq COEF19 5.569089e-05 0.32228 !*002004
ueq COEF20 3.303353e-06 0.22565 !*002005
ueq COEF21 1.574620e-04 0.20425 !*002006
ueq COEF22 1.766409e-03 0.23673 !*003004
ueq COEF23 2.299110e-04 0.17680 !*003005
ueq COEF24 1.658469e-03 0.20599 !*003006
ueq COEF25 -2.399434e-04 0.00106 !*004005
ueq COEF26 -1.575718e-02 0.17891 !*004006
ueq COEF27 -1.076411e-03 0.26856 !*005006
REM The preceding lines are easily installed in this file
REM by extracting lines from the F output file and editing
REM them to put in the characters "ueq COEF".

```

```

; measurement value extrema:

```

```

udefF32 S1MIN S2MIN S3MIN S4MIN S5MIN S6MIN S7MIN
ueq S1MIN -300000. -270801.
ueq S2MIN 0.0 6472.7
ueq S3MIN -300. -201.8
ueq S4MIN -10. -9.
ueq S5MIN -1000. -807.1
ueq S6MIN -80. -57.
ueq S7MIN -30. -21.9

```

```

udefI32 NCOL NROW NOVL ; scene image size

```

```

ueq NCOL 43
ueq NROW 25
ueq NOVL 2

```

```

udefI32 RESULT BIG1 BIG2 ; image numbers

```

```

ueq RESULT 8 ; result image
ueq BIG1 9
ueq BIG2 10

```

```

udefI32 SI1 SI2 SI3 SI4 SI5 SI6 SKR1 ; more image numbers

```

```

ueq SI1 1
ueq SI2 2
ueq SI3 3
ueq SI4 4
ueq SI5 5
ueq SI6 6
ueq SKR1 7

```

```

; input all scene images, reading and using scale factors:

```

```

readscene SI1
subcon SI1 SI1 $A

```

```

divcon SI1 SI1 $B
subcon SI1 SI1 S1MIN
readscene SI2
subcon SI2 SI2 $A
divcon SI2 SI2 $B
subcon SI2 SI2 S2MIN
readscene SI3
subcon SI3 SI3 $A
divcon SI3 SI3 $B
subcon SI3 SI3 S3MIN
readscene SI4
subcon SI4 SI4 $A
divcon SI4 SI4 $B
subcon SI4 SI4 S4MIN
readscene SI5
subcon SI5 SI5 $A
divcon SI5 SI5 $B
subcon SI5 SI5 S5MIN
readscene SI6
subcon SI6 SI6 $A
divcon SI6 SI6 $B
subcon SI6 SI6 S6MIN

```

```

copdef RESULT SI1 0
copdef SKR1 SI1 NOVL

```

```

; use constant feature, feature 0:
zeroimage RESULT
addcon RESULT RESULT COEF0

```

```

; use absolute magnitude as a feature:
mulcon SKR1 SI1 COEF1
add RESULT RESULT SKR1
mulcon SKR1 SI2 COEF2
add RESULT RESULT SKR1
mulcon SKR1 SI3 COEF3
add RESULT RESULT SKR1
mulcon SKR1 SI4 COEF4
add RESULT RESULT SKR1
mulcon SKR1 SI5 COEF5
add RESULT RESULT SKR1
mulcon SKR1 SI6 COEF6
add RESULT RESULT SKR1
;use quadratic features:
mul SKR1 SI1 SI1
mulcon SKR1 SKR1 COEF7
add RESULT RESULT SKR1
mul SKR1 SI2 SI2
mulcon SKR1 SKR1 COEF8
add RESULT RESULT SKR1
mul SKR1 SI3 SI3

```

```

mulcon SKR1 SKR1 COEF9
add RESULT RESULT SKR1
mul SKR1 SI4 SI4
mulcon SKR1 SKR1 COEF10
add RESULT RESULT SKR1
mul SKR1 SI5 SI5
mulcon SKR1 SKR1 COEF11
add RESULT RESULT SKR1
mul SKR1 SI6 SI6
mulcon SKR1 SKR1 COEF12
add RESULT RESULT SKR1
mul SKR1 SI1 SI2
mulcon SKR1 SKR1 COEF13
add RESULT RESULT SKR1
mul SKR1 SI1 SI3
mulcon SKR1 SKR1 COEF14
add RESULT RESULT SKR1
mul SKR1 SI1 SI4
mulcon SKR1 SKR1 COEF15
add RESULT RESULT SKR1
mul SKR1 SI1 SI5
mulcon SKR1 SKR1 COEF16
add RESULT RESULT SKR1
mul SKR1 SI1 SI6
mulcon SKR1 SKR1 COEF17
add RESULT RESULT SKR1
mul SKR1 SI2 SI3
mulcon SKR1 SKR1 COEF18
add RESULT RESULT SKR1
mul SKR1 SI2 SI4
mulcon SKR1 SKR1 COEF19
add RESULT RESULT SKR1
mul SKR1 SI2 SI5
mulcon SKR1 SKR1 COEF20
add RESULT RESULT SKR1
mul SKR1 SI2 SI6
mulcon SKR1 SKR1 COEF21
add RESULT RESULT SKR1
mul SKR1 SI3 SI4
mulcon SKR1 SKR1 COEF22
add RESULT RESULT SKR1
mul SKR1 SI3 SI5
mulcon SKR1 SKR1 COEF23
add RESULT RESULT SKR1
mul SKR1 SI3 SI6
mulcon SKR1 SKR1 COEF24
add RESULT RESULT SKR1
mul SKR1 SI4 SI5
mulcon SKR1 SKR1 COEF25
add RESULT RESULT SKR1
mul SKR1 SI4 SI6

```



```

mulcon SKR1 SKR1 COEF26
add RESULT RESULT SKR1
mul SKR1 SI5 SI6
mulcon SKR1 SKR1 COEF27
add RESULT RESULT SKR1

mincon RESULT RESULT 255.0
maxcon RESULT RESULT 0.0

writeresult RESULT 1 Q

STOP

; display result:
resample BIG1 RESULT 0 100 100 0
TTG3 BIG1 50 50 Q

copdef BIG2 BIG1 0
resample BIG2 SI1 0 100 100 0
scale BIG2 BIG2
echo_on
TTG3RES BIG2 50 200 Q BIG1

stop

```

Running E with these files generates 30 result images. These can be combined into one composite image for display, as in the figures printed here, with the following E program commands:

Program E command file:

```

0
e4i2.sl
e4i2.fc
NUL
look for barrels

```

Program E scene list file:

```

1 scene
30 images
\ri\com4i2.img
\ri\E2L0.IMG      1587 02-24-94   3:47p
\ri\E2L1.IMG      1587 02-24-94   3:47p
.
.   FILE TEXT IS MISSING HERE
.

```

This scene list file was conveniently constructed by editing a directory listing. The file size and date information, being separated from the file name by white space, is ignored by the

ATR program.

Program E feature command file:

```
; collect and compose results from e4i.
echo_off
undefI32 LEVELS EXPS
    ueq LEVELS 9
    ueq EXPS 4
undefI32 INN BIG1 BIG3
undefI32 NCOL0 NROW0 NCOL1 NROW1
    ueq NCOL0 43 ; size of raw result image
    ueq NROW0 25
    ueq NCOL1 43 ; size of displayed result image
    ueq NROW1 50
undefI32 ROW0 COL0 DELROW DELCOL ROW COL ; screen coordinates etc.
    ueq ROW0 0
    ueq COL0 0
    ueq INN 1 ; image number, raw result image
    ueq BIG1 11 ; image number, displayed result image
    ueq BIG3 13 ; image number, composite image
undefF32 Q

; defimg INN NCOL0 NROW0 0
; defimg BIG1 NCOL1 NROW1 0
uadd DELCOL NCOL1 3 ; allow 3 columns
uadd DELROW NROW1 6 ; and 6 rows between images
umul COL DELCOL LEVELS
umul ROW DELROW EXPS
defimg BIG3 COL ROW 0
zeroimage BIG3
addcon BIG3 BIG3 255.1 ; white background

; row 1, experiment 2, 9 levels:
ueq ROW ROW0
ueq COL COL0

readscene INN
    ueq Q $A ; get Q value
resample BIG1 INN 0 NCOL1 NROW1 0
compoz BIG3 ROW COL BIG1
uadd COL COL DELCOL

readscene INN
resample BIG1 INN 0 NCOL1 NROW1 0
compoz BIG3 ROW COL BIG1
uadd COL COL DELCOL

readscene INN
resample BIG1 INN 0 NCOL1 NROW1 0
```

compoz BIG3 ROW COL BIG1
uadd COL COL DELCOL

readscene INN
resample BIG1 INN 0 NCOL1 NROW1 0
compoz BIG3 ROW COL BIG1
uadd COL COL DELCOL

readscene INN
resample BIG1 INN 0 NCOL1 NROW1 0
compoz BIG3 ROW COL BIG1
uadd COL COL DELCOL

readscene INN
resample BIG1 INN 0 NCOL1 NROW1 0
compoz BIG3 ROW COL BIG1
uadd COL COL DELCOL

readscene INN
resample BIG1 INN 0 NCOL1 NROW1 0
compoz BIG3 ROW COL BIG1
uadd COL COL DELCOL

readscene INN
resample BIG1 INN 0 NCOL1 NROW1 0
compoz BIG3 ROW COL BIG1
uadd COL COL DELCOL

readscene INN
resample BIG1 INN 0 NCOL1 NROW1 0
compoz BIG3 ROW COL BIG1
uadd COL COL DELCOL

; row 2, experiment 3, 8 levels:
uadd ROW ROW DELROW
ueq COL COLO

readscene INN
resample BIG1 INN 0 NCOL1 NROW1 0
compoz BIG3 ROW COL BIG1
uadd COL COL DELCOL

.
FILE TEXT IS MISSING HERE
.

writeresult BIG3 1 Q ; write composite image
TTG3 BIG3 100 100 q ; display composite image

stop

The same quadratic analysis procedure was used to try to find boxes instead of barrels. In this analysis, the training process used two training scenes, E5L5 with mask M5C and E2L7 with mask M2B. These masks select only boxes without ferrous metal contents as targets. The results of this analysis for 30 scenes are shown in Figure C5. This analysis was repeated using only the first training scene, with results shown in Figure C6. Obviously, the results of this type of analysis can depend significantly on the choice of training data. In both of these two analyses, boxes (objects 6, 9, 10, 21, 22, 23, and 24) were correctly identified as targets, and a few other objects were incorrectly identified as targets, with the single-training-scene result a little better than the two-training-scene result. This result is reasonably good.

The files needed for doing this quadratic analysis for boxes are almost the same as those used for the quadratic analysis for barrels. Only the F scene list file and the coefficient values are different.

An interesting aspect of these linear and quadratic analyses is that they require no understanding of the measurements, except for a knowledge of the minimum numerical values for each of the 6 measurement types.

In an effort to more accurately identify barrels, we tried a more sophisticated analysis that made use of some understanding of the measurement processes. In this analysis, we made an effort to find peaks in the magnetic field and its gradient, and to discriminate on the basis of the widths of the peaks, on the assumption that barrels should cause peaks of a certain width in magnetic field measurements. This analysis used only sensors S1B and S1A; the other sensor data was ignored. The training process used the same training scenes as were used for the quadratic function search for barrels. The results of this analysis are shown in Figure C7. Note that experiment 1 is included, since this analysis does not require the sensors that are not included in experiment 1. This analysis was repeated with a different set of training data, using scenes E2L8, E5L2, E5L5, and E6L6 with the same masks as before. This gave slightly better results, shown in Figure C8. These results generally show the barrels as targets, which is good. They also show other ferromagnetic objects as targets, which we had hoped to avoid. However, it is not surprising that we are not able to distinguish well between barrels and other magnetic objects, since the peak width which we tried as the distinguishing feature is not really unique to barrels. Surprisingly, this analysis also shows as targets some presumed non-magnetic objects, such as a box (object 6) in E2. This is interpreted as a fairly strong indication that these boxes do in fact include some ferromagnetic material. Listings of some files relevant to this analysis follow:

F program feature command file:

```
; look for barrels.  
; try finding peaks and their widths.
```

```
udefF32 S1ZERO S2ZERO  
    ueq S1ZERO    -1000.  
    ueq S2ZERO    51000.
```

```

udefI32 NCOL NROW NOVL
    ueq NCOL 43
    ueq NROW 25
    ueq NOVL 2
udefI32 BIGMASK MASK SCENE SKR1
    ueq BIGMASK 1
    ueq MASK 2
    ueq SKR1 2
    ueq SCENE 3
udefI32 C1 CU CUU WIDTH WEIGHT
    ueq C1 4
    ueq CU 5
    ueq CUU 6
    ueq WIDTH 7
    ueq WEIGHT 1

defkern WEIGHT -2 2 -2 2
readkernel WEIGHT
    \imanal\opr\opr21a\mask.opr

; input, resample (resize), and output mask:
readscene BIGMASK
resample MASK BIGMASK 0 NCOL NROW 0
writefeat MASK ; feature 0
    d:MASK.img
defimg BIGMASK 0 0 0
defimg MASK 0 0 0

defimg SCENE 0 0 0
readscene SCENE
subcon SCENE SCENE $A
divcon SCENE SCENE $B
subcon SCENE SCENE $ZERO

copdef SKR1 SCENE NOVL
copdef C1 SCENE 0
copdef CU SCENE 0
copdef CUU SCENE 0
copdef WIDTH SCENE 0

copy SKR1 SCENE
overlap SKR1
quaduv SKR1 WEIGHT C1 CU 0 CUU 0 0 ; quadratic function fit
maxcon SKR1 C1 0.0
overlap SKR1
median C1 SKR1 WEIGHT 3 3 ; positive z only
copy SKR1 CU
overlap SKR1
median CU SKR1 WEIGHT 3 3 ; smoothed linear coefficient
mulcon SKR1 CUU -1.0

```

```

overlap SKR1
median CUU SKR1 WEIGHT 3 3
maxcon CUU CUU 0.0 ; dome
mul WIDTH CU CU
div WIDTH WIDTH C1
div WIDTH WIDTH CUU
addcon WIDTH WIDTH 4.0
mul WIDTH WIDTH C1
div WIDTH WIDTH CUU
rephi SKR1 WIDTH 1.0e5 0.0
overlap SKR1
median WIDTH SKR1 WEIGHT 0 4
sqrt WIDTH WIDTH

```

```

scale CUU CUU
writefeat CUU
  d:DOMEl.img
scale WIDTH WIDTH
writefeat WIDTH
  d:WIDTH1.img
feat* 2 2
feat* 1 2
feat* 1 3

```

```

defimg SCENE 0 0 0
readscene SCENE
subcon SCENE SCENE $A
divcon SCENE SCENE $B
subcon SCENE SCENE S2ZERO
copy SKR1 SCENE
overlap SKR1
quaduv SKR1 WEIGHT C1 CU 0 CUU 0 0 ; quadratic function fit
maxcon SKR1 C1 0.0
overlap SKR1
median C1 SKR1 WEIGHT 3 3 ; positive z only
copy SKR1 CU
overlap SKR1
median CU SKR1 WEIGHT 3 3 ; smoothed linear coefficient
mulcon SKR1 CUU -1.0
overlap SKR1
median CUU SKR1 WEIGHT 3 3
maxcon CUU CUU 0.0 ; dome
mul WIDTH CU CU
div WIDTH WIDTH C1
div WIDTH WIDTH CUU
addcon WIDTH WIDTH 4.0
mul WIDTH WIDTH C1
div WIDTH WIDTH CUU
rephi SKR1 WIDTH 1.0e5 0.0
overlap SKR1

```

```
median WIDTH SKR1 WEIGHT 0 4
sqrt WIDTH WIDTH
```

```
scale CUU CUU
writefeat CUU
  d:DOME2.img
scale WIDTH WIDTH
writefeat WIDTH
  d:WIDTH2.img
feat* 7 7
feat* 6 7
feat* 6 3
```

```
;skip over unused scene images:
;(this is very inefficient, not recommended!)
;(change the scene list file, you lazy toad!)
readscene SCENE
readscene SCENE
readscene SCENE
readscene SCENE
```

```
stop
```

E program feature command file:

```
; look for barrels.
; find peaks and their widths.
```

```
echo_off
```

```
undefF32 Q COEF0 COEF1 COEF2 COEF3 COEF4 COEF5 COEF6
undefF32 COEF7 COEF8 COEF9 COEF10
```

```
  ueq Q 95.624916 = Q = target criterion.
  ueq COEF0 5.861998e+01 0.19509 d:MASK.img
  ueq COEF1 -5.521925e-02 0.19809 d:DOME1.img
  ueq COEF2 6.935584e+00 0.24768 d:WIDTH1.img
  ueq COEF3 -1.539143e-01 0.00885 !*002002
  ueq COEF4 1.111876e-02 0.34271 !*001002
  ueq COEF5 -3.884335e-04 0.07713 !*001003
  ueq COEF6 1.896816e-01 0.33799 d:DOME2.img
  ueq COEF7 5.446849e+00 0.24136 d:WIDTH2.img
  ueq COEF8 -1.921345e-01 0.05092 !*007007
  ueq COEF9 -6.553839e-03 0.00110 !*006007
  ueq COEF10 -2.992128e-03 0.07640 !*006003
```

```
undefF32 S1ZERO S2ZERO
  ueq S1ZERO -1000.
  ueq S2ZERO 51000.
```

```
undefI32 NCOL NROW NOVL
  ueq NCOL 43
```

```

    ueq NROW 25
    ueq NOVL 2
    udefi32 SCENE SKR1
    ueq SKR1 2
    ueq SCENE 3
    udefi32 C1 CU CUU WIDTH WEIGHT
    ueq C1 4
    ueq CU 5
    ueq CUU 6
    ueq WIDTH 7
    ueq WEIGHT 1
    udefi32 RESULT BIG1 BIG2
    ueq RESULT 8
    ueq BIG1 9
    ueq BIG2 10

```

```

defkern WEIGHT -2 2 -2 2
readkernel WEIGHT
\imanal\opr\opr21a\mask.opr

```

```

defimg SCENE 0 0 0
readscene SCENE
subcon SCENE SCENE $A
divcon SCENE SCENE $B
subcon SCENE SCENE SIZERO
    copdef SKR1    SCENE NOVL
    copdef C1      SCENE 0
    copdef CU      SCENE 0
    copdef CUU     SCENE 0
    copdef WIDTH   SCENE 0
    copdef RESULT  SCENE 0
    zeroimage RESULT
    ; use constant feature:
    addcon RESULT RESULT COEFO
copy SKR1 SCENE
overlap SKR1
quaduv SKR1 WEIGHT C1 CU 0 CUU 0 0 ; quadratic function fit
maxcon SKR1 C1 0.0
overlap SKR1
median C1 SKR1 WEIGHT 3 3 ; positive z only
copy SKR1 CU
overlap SKR1
median CU SKR1 WEIGHT 3 3 ; smoothed linear coefficient
mulcon SKR1 CUU -1.0
overlap SKR1
median CUU SKR1 WEIGHT 3 3
maxcon CUU CUU 0.0 ; dome
mul WIDTH CU CU
div WIDTH WIDTH C1
div WIDTH WIDTH CUU

```



```

addcon WIDTH WIDTH 4.0
mul WIDTH WIDTH C1
div WIDTH WIDTH CUU
rephi SKR1 WIDTH 1.0e5 0.0
overlap SKR1
median WIDTH SKR1 WEIGHT 0 4
sqrt WIDTH WIDTH ; width
mulcon SKR1 CUU COEF1 ; dome
add RESULT RESULT SKR1
mulcon SKR1 WIDTH COEF2 ; width
add RESULT RESULT SKR1
mul SKR1 SKR1 SKR1
mulcon SKR1 SKR1 COEF3 ; width*width
add RESULT RESULT SKR1
mul SKR1 SKR1 CUU
mulcon SKR1 SKR1 COEF5 ; dome*width*width
add RESULT RESULT SKR1
mul SKR1 CUU WIDTH
mulcon SKR1 SKR1 COEF4 ; dome*width
add RESULT RESULT SKR1

```

```

defimg SCENE 0 0 0
readscene SCENE
subcon SCENE SCENE $A
divcon SCENE SCENE $B
subcon SCENE SCENE S2ZERO
copy SKR1 SCENE
overlap SKR1
quaduv SKR1 WEIGHT C1 CU 0 CUU 0 0 ; quadratic function fit
maxcon SKR1 C1 0.0
overlap SKR1
median C1 SKR1 WEIGHT 3 3 ; positive z only
copy SKR1 CU
overlap SKR1
median CU SKR1 WEIGHT 3 3 ; smoothed linear coefficient
mulcon SKR1 CUU -1.0
overlap SKR1
median CUU SKR1 WEIGHT 3 3
maxcon CUU CUU 0.0 ; dome
mul WIDTH CU CU
div WIDTH WIDTH C1
div WIDTH WIDTH CUU
addcon WIDTH WIDTH 4.0
mul WIDTH WIDTH C1
div WIDTH WIDTH CUU
rephi SKR1 WIDTH 1.0e5 0.0
overlap SKR1
median WIDTH SKR1 WEIGHT 0 4
sqrt WIDTH WIDTH ; width
mulcon SKR1 CUU COEF6 ; dome

```

```

add RESULT RESULT SKR1
mulcon SKR1 WIDTH COEF7 ; width
add RESULT RESULT SKR1
mul SKR1 SKR1 SKR1
mulcon SKR1 SKR1 COEF8 ; width*width
add RESULT RESULT SKR1
mul SKR1 SKR1 CUU
mulcon SKR1 SKR1 COEF10 ; dome*width*width
add RESULT RESULT SKR1
mul SKR1 CUU WIDTH
mulcon SKR1 SKR1 COEF9 ; dome*width
add RESULT RESULT SKR1

```

```

readscene SCENE
readscene SCENE
readscene SCENE
readscene SCENE

```

```

mincon RESULT RESULT 255.0
maxcon RESULT RESULT 0.0

```

```

writeresult RESULT 1 Q

```

STOP

```

; display result:
resample BIG1 RESULT 0 100 100 0
TTG3 BIG1 50 50 Q

```

STOP

```

copdef BIG2 BIG1 0
resample BIG2 SI1 0 100 100 0
scale BIG2 BIG2
echo_on
TTG3RES BIG2 50 200 Q BIG1

```

stop

Kernel file \imanal\opr\opr21a\mask.opr:

0.00000	1.0	1.0	1.0	0.00000
1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0
0.00000	1.0	1.0	1.0	0.00000

APPENDIX D

EXAMPLE 4: Finding Airplanes

EXAMPLE 4: Finding Airplanes

This example indicates some techniques for using the ATR system, and it includes an instance of a feature being classified as unimportant.

We start with a fairly simple training procedure. The feature calculation file is flb.fc:

```
; Find airplanes.

ECHO flb.fc

sdefI32 NOVL
seq NOVL 7
sdefI32 RAW SMOOTH RUFF CROSS TEMP ; image numbers
seq RAW 1
seq SMOOTH 2
seq RUFF 3
seq CROSS 4
seq TEMP 5
defimg RAW 512 512 NOVL ; raw scene image
copdef SMOOTH RAW ; smoothed
copdef RUFF RAW NOVL ; smoothed - raw
copdef CROSS RAW ; crosses
copdef TEMP RAW ; scratch

sdefI32 KERNEL1 ITEM
seq KERNEL1 1
ssub ITEM 0 NOVL
defkern KERNEL1 ITEM NOVL ITEM NOVL

featfil ; use scene mask for feature mask

readscene RAW 1 0 0 0

sdefF32 SIG1
seq SIG1 7.0 ; smoothing width, pass 1
smthx SMOOTH RAW SIG1 5 0
smthy SMOOTH SMOOTH SIG1 5 0
writefeat SMOOTH ; already close enough to scaled
SCSIDISK:EX3\SMOOTHED

sub RUFF SMOOTH RAW
maxcon RUFF RUFF 0.0 ; keep only dark spots of raw image
scale TEMP RUFF 0
writefeat TEMP
SCSIDISK:EX3\RUFF

feat* 1 2

readkernel KERNEL1
C:\imanal\opr\scopr\c4t15.opr
```

```

convolve TEMP RUFF KERNEL1
mul CROSS TEMP TEMP
readkernel KERNEL1
  C:\imanal\opr\scopr\s4t15.opr
convolve TEMP RUFF KERNEL1
mul TEMP TEMP TEMP
add CROSS CROSS TEMP
scale TEMP CROSS KERNEL1
writefeat TEMP ; 4*theta, crosses
  SCSIDISK:EX3\SC4

```

STOP

This calculates and uses as features a smoothed scene image (SMOOTH), a feature image that indicates how much darker a small local region is than the smoothed image (RUFF), their product, and the indicator of cross patterns in the RUFF image (SC4). The calculation of the first three feature images is quite obvious, but the last may bear a little explanation. The two kernels c4t15.opr and s4t15.opr are both zero in the centers and in the corners, being non-zero only in a circular annular region that is intended to overlap the wings and nose and tail of an airplane image but not the center of the airplane. The non-zero elements of the kernels are proportional to the cosine and sine respectively of four times the angle from the x axis. Thus, when convolved with an image, the cosine kernel will indicate the presence of four-lobed patterns with the lobes aligned with the x and y axes, and the sine kernel will indicate those rotated by 45 degrees, and the sum of the squares will indicate the square of the magnitude of the four-lobed pattern independent of its orientation. That is, this procedure finds cross-shaped patterns. It is, of course, also sensitive to certain other patterns, including straight lines.

The relevant part of the training program output file flb.out is:

RESTART:

```

for 5 active features: 0.008626
0.008626 = 0.980 * 0.001352 + 0.020 * 0.365078
Q = 118.414795 = target criterion.

f      C[f]      import.
0  6.563824e+01  0.00158 c:\imanal\cims\ex3\maskd.img
1 -3.993893e-02  0.00006 SCSIDISK:EX3\SMOOTHED
2 -3.119725e+00  0.00650 SCSIDISK:EX3\RUFF
3  3.075633e-02  0.14078 !*001002
4  1.127659e-03  0.00001 SCSIDISK:EX3\SC4

```

The significant quantity here is the very low value, 0.00001, for the importance of the crosses feature. This indicates that it would not make any difference if this feature were not included. The SMOOTHED feature also has a low importance. However, we cannot conclude from this information alone that it is safe to remove both the SC4 and the SMOOTHED features; removing either one might greatly increase the importance of the

other. In this example, the importance of the SMOOTHED feature after the SC4 feature is deleted is indicated by the next part of the output file:

```
for 4 active features: 0.008633
0.008633 = 0.980 * 0.001353 + 0.020 * 0.365364
Q = 118.561653 = target criterion.

f      C[f]      import.
0 6.650915e+01 0.00158 c:\imanal\cims\ex3\maskd.img
1 -4.337708e-02 0.00007 SCSIDISK:EX3\SMOOTHED
2 -3.185944e+00 0.00554 SCSIDISK:EX3\RUFF
3 3.145166e-02 0.07333 !*001002
```

This indicates that the SMOOTHED feature is still not important, even after the removal of the SC4 feature, so the SMOOTHED feature could be removed without significantly affecting the accuracy of the target recognition process.

Imposing stringent size constraints on the non-zero areas of the RUFF image that are candidates for targets is not a trivial procedure. The process used here is to treat local regions of the RUFF image as if the intensity versus x and y were actually an un-normalized probability density function (distribution), and to use the two principal moments of each peak in the distribution as indicators of the width of the peak. Experimentation gives the values of the maximum and minimum acceptable widths. Implementation of this procedure is shown in the listing of the surveillance feature calculation file elb.fc:

ECHO elb.fc

```
sdefF32 Q COEF0 COEF1 COEF2 COEF3 COEF4 COEF5 COEF6 COEF7 COEF8
seq COEF0 0.0
seq COEF1 0.0
seq COEF2 0.0
seq COEF3 0.0
seq COEF4 0.0
seq COEF5 0.0
seq COEF6 0.0
seq COEF7 0.0
seq COEF8 0.0
seq Q 118.561653 = target criterion.
seq COEF0 6.650915e+01 0.00158 c:\imanal\cims\ex3\maskd.img
seq COEF1 -4.337708e-02 0.00007 SCSIDISK:EX3\SMOOTHED
seq COEF2 -3.185944e+00 0.00554 SCSIDISK:EX3\RUFF
seq COEF3 3.145166e-02 0.07333 !*001002

sdefI32 NOVL
seq NOVL 7
defimg 1 512 512 NOVL
copdef 2 1 0
copdef 3 1 NOVL
copdef 4 1 0
copdef 5 1 0
```

```

copdef 6 1 0
copdef 7 1 0
copdef 8 1 NOVL
copdef 9 1 NOVL

sdefI32 KERNEL1 TEMP
seq KERNEL1 1
ssub TEMP 0 NOVL
defkern KERNEL1 TEMP NOVL TEMP NOVL

sdefI32 RESULT
seq RESULT 9
zeroimage RESULT
addcon RESULT RESULT COEF0
seq TEMP 8

sdefI32 RAW ; raw scene image
seq RAW 1
readscene RAW 1 0 0 0
TTG3 RAW 0 767 0.0
    RAW_SCENE

sdefF32 SIG
seq SIG 7.0 ; smoothing width, pass 1
sdefI32 SMOOTH ; smoothed scene image
seq SMOOTH 2
smthx SMOOTH RAW SIG 5 0
smthy SMOOTH SMOOTH SIG 5 0
mulcon TEMP SMOOTH COEF1
add RESULT RESULT TEMP

sdefI32 RUFF ; rough part, smoothed-raw
seq RUFF 3
sub RUFF SMOOTH RAW
maxcon RUFF RUFF 0.0 ; keep only dark spots of raw image
mulcon TEMP RUFF COEF2
add RESULT RESULT TEMP

mul TEMP SMOOTH RUFF
mulcon TEMP TEMP COEF3
add RESULT RESULT TEMP

mul TEMP RUFF RUFF
mulcon TEMP TEMP COEF4
add RESULT RESULT TEMP

mul TEMP SMOOTH SMOOTH
mulcon TEMP TEMP COEF5
add RESULT RESULT TEMP

readkernel KERNEL1

```

```

C:\imanal\opr\scopr\c4t15.opr
convolve TEMP RUFF KERNEL1
mul CROSS TEMP TEMP
readkernel KERNEL1
C:\imanal\opr\scopr\s4t15.opr
convolve TEMP RUFF KERNEL1
mul TEMP TEMP TEMP
add CROSS CROSS TEMP
mulcon TEMP CROSS COEF6
add RESULT RESULT TEMP

```

```

maxcon RESULT RESULT 0.0
mincon RESULT RESULT 255.0
;writeresult RESULT 1 Q
TTG3 RESULT 0 255 Q
RESULT,NO_MOMUV_MASK
TTG3res RAW 512 255 Q RESULT
RESULT,NO_MOMUV_MASK

```

;Calculate peak width limit mask:

```

readkernel KERNEL1
C:\imanal\opr\scopr\avg15.opr

```

```

sdefI32 MOMU MOMV MOMUU MOMVV ; moments of PDF
seq MOMU 7
seq MOMV 6
seq MOMUU 5
seq MOMVV 4
overlap RUFF
momuv RUFF KERNEL1 0 MOMU MOMV MOMUU MOMVV
; convert from second moment about 0 to variance:
seq TEMP 8
mul TEMP MOMU MOMU
sub MOMUU MOMUU TEMP
mul TEMP MOMV MOMV
sub MOMVV MOMVV TEMP

```

```

sdefF32 MAXMANDIST ; maximum Manhattan distance to centroid
seq MAXMANDIST 1.0
REM MAXMANDIST should be at least 0.9
sdefI32 MANDIST ; Manhattan distance to centroid of PDF
seq MANDIST 8
abs MOMU MOMU
abs MOMV MOMV
max MANDIST MOMU MOMV ; not really Manhattan distance
rephi MANDIST MANDIST MAXMANDIST -1.0
rephi MANDIST MANDIST 0.01 1.0
replo MANDIST MANDIST 0.5 0.0

```



```

mul  MOMUU MOMUU MANDIST
mul  MOMVV MOMVV MANDIST

sdefF32  MAXMOMUU ; maximum allowable UU moment
sdiv  MAXMOMUU NOVL 2.1 ; max sigma <= NOVL / sqrt(3)
REM  Trials indicate that this number should be <= 2.1 for NOVL = 7.
smul  MAXMOMUU MAXMOMUU MAXMOMUU
rephi  MOMUU MOMUU MAXMOMUU 0.0

sdefF32  MINMOMUU ; minimum UU moment
smul  MINMOMUU 2.7 2.7
REM  Trials indicate that this number should be >= 2.7 .
rephi  MOMUU MOMUU MINMOMUU 0.0

sdefI32  RATIO ; MOMVV/MOMUU
seq  RATIO MOMVV
div  RATIO MOMVV MOMUU
rephi  RATIO RATIO 2.00 0.0 ; eliminate division by 0 effects

sdefF32  MINRATIO
seq  MINRATIO 0.4
rephi  RATIO RATIO MINRATIO 0.0
rephi  RATIO RATIO 0.01 1.0
; image RATIO is a mask, values either 1.0 or 0.0 .

; broaden non-zero mask regions:
copy  TEMP RATIO
overlap  TEMP
convolve  RATIO TEMP KERNEL1
rephi  RATIO RATIO 0.0001 1.0

; apply mask:
mul  RESULT RESULT RATIO
; image RESULT is a standard type 1 result image.

; delete too-small target indications:
seq  SIG 5.0
rephi  TEMP RESULT Q 0.0
rephi  TEMP TEMP 1.0 255.0 ; binary, 0 or 255.
smthx  TEMP TEMP SIG 5
smthy  TEMP TEMP SIG 5
rephi  TEMP TEMP 45.0 0.0
rephi  TEMP TEMP 1.0 1.0
mul  RESULT RESULT TEMP

; Image RESULT is standard type 1 result file.
;writeresult  RESULT 1 Q
TTG3  RESULT 0 767 Q
      RESULT\MOMUV_MASK

```

TTG3res RAW 512 767 Q RESULT

```
; Standard type 2 result file sequence:
; draw loops around targets:
sdefI32 RESULT2
seq RESULT2 3
seq SIG 5.0 ; minimum loop radius [pixels]
replo RESULT2 RESULT Q 0.0
rephi RESULT2 RESULT2 1.0 255.0 ; binary, 0 or 255.
smthx RESULT2 RESULT2 SIG 5
smthy RESULT2 RESULT2 SIG 5
smul SIG SIG SIG
sdiv SIG 24.61575 SIG
zeroimage TEMP
overlap RESULT2 ; RESULT2 should have at least 1 overlap row!
contour TEMP RESULT2 SIG 255.0 ; loops 1 pixel wide.
defkern KERNEL1 -1 1 -1 1
readkernel KERNEL1 ; kernel is 3x3, all pixel values are 1.
c:\imanal\cims\med3i.opr
overlap TEMP ; TEMP should have at least 1 overlap row!
convolve RESULT2 TEMP KERNEL1
rephi RESULT2 RESULT2 1.0 255.0 ; binary, 3-pix-wide loops.
sub RESULT2 RESULT2 TEMP ; binary, double 1-pix-wide loops.
;writeimage RESULT2 1
; LOOPS.img
;TTG3 RESULT2 0 767 0.0
; .
sub TEMP RAW TEMP
maxcon TEMP TEMP 0.0 ; scene with black 1-pix-wide loops.
mincon TEMP TEMP 254.0
max RESULT2 RESULT2 TEMP ; scene with white-black-white loops.
writeresult RESULT2 1 Q
TTG3 RESULT2 512 767 254.0 ; standard type 2 result display.
```

STOP

This feature calculation file includes a lot of diagnostic and experimental stuff that would be removed from a real production feature calculation file, and it shows a method for drawing circles around targets to make them very noticable to a human analyst.

APPENDIX E

E9 AND F9 FEATURE CALCULATION COMMANDS

E9 AND F9 FEATURE CALCULATION COMMANDS

CONTROL OF PROGRAM

REM, ECHO, ECHO_ON, ECHO_OFF, ECHOTIME, STOP,
LABEL, JUMP, BRANCH, IF, END

DEFINING IMAGES AND KERNELS (OPERATORS)

DEFIMG, DEFKERN, COPDEF, CLEAR; RESAMPLE, UNDERSAMPLE,
READIMAGE, READSCENE

READING IMAGES AND KERNELS FROM FILES

READIMAGE, READSCENE, READKERNEL

BASIC IMAGE OPERATIONS

ZEROIMAGE, SETPIX, NTRP00, NTRP01, COPY, COPYEDGE, OVERLAP,
RESAMPLE, UNDERSAMPLE, INSERT, EXTRACT

ARITHMETIC OPERATIONS ON IMAGES

MAX, MIN, MAXCON, MINCON, REPLO, REPHI, ADD, SUB, MUL, DIV, ADDCON,
SUBCON, MULCON, DIVCON, SQRT, ABS: All the images in these operations must be
already defined, all the same size, and all distributed the same way among the daisies. The
images need not be different; dst can be the same as src or src2, etc. const is a single
floating point value. These operations do not use or set the values of overlap rows.

CONVOLUTION and related operations

CONVOLVE, SSQ

GRADIENTS

GRADT, XGRAD, YGRAD, GRADCON; XY2RT

FILTERING IMAGES

SMTHX, SMTHY, MEDIAN, MED1X, MED1Y

MISCELLANEOUS IMAGE OPERATIONS

LIN01, LIN02, XLIN01, NLIN01, EDJ02

MODMSK

QUADXY, QUADUV

SEGLAB

RIJCON, GRADCON

MOMUV

DISPLAYING IMAGES

VGA, TTG3, TTG3RES; SCALE, CONTOUR

IMAGE INFORMATION

STATS, HIST2; SET_AB

FEATURE IMAGES

WRITEFEAT, FEATFIL, FEAT*; SCALE

WRITING IMAGES TO FILES

WRITEIMAGE, WRITEFEAT, WRITERESULT; SCALE

SCALAR VARIABLES

PRINTS, SABS, SADD, SDEFF32, SDEFI32, SDIV, SEQ, SMAX, SMIN, SMUL, SSQRT, SSUB (formerly called PRINTU, UABS, UADD, UDEFF32, UDEFI32, UDIV, UEQ, UMAX, UMIN, UMUL, USQRT, USUB)

The user can define scalar (not array or image) variables, either 32-bit integers or 32-bit floating, set their values, print their values, and use them as parameters in commands. Some scalar variables are pre-defined and their values are automatically set by certain operations. The pre-defined scalar variables are named \$N, \$NCOL, \$NROW (integers), and \$AVG, \$SIG, \$MAX, \$MIN, \$A, and \$B (floating). The program does not distinguish between upper and lower case characters in variable names. The arithmetic operations for scalar variables allow free mixing of integer and floating types.

ABS dst, src

Set image **dst** = absolute value of image **src**.

ADD dst, src1, src2

Set image **dst** = image **src1** + image **src2**.

ADDCON dst, src, const

Set image **dst** = image **src** + **const**.

BRANCH n

Jump to the .fc file line which is **n** lines removed from the line containing this **BRANCH** statement. **n** may be positive or negative. **n=0** is an error condition, since this would establish an infinite loop of repeatedly branching to the current line. Although **n=1** is not an error, a **BRANCH** command with **n=1** is superfluous. It causes a jump to the following line, which would be the normal sequence even if the **BRANCH** statement were absent.

CLEAR (no parameters)

Clear the tables that hold information about images, operators, and user-defined scalar variables, and free the memory allocated for these things.

CONTOUR dst, src, sval, dval

This operation draws one contour curve in destination image **dst**, representing a curve of constant intensity in source image **src**. If a pixel in **src** has intensity at least as great as the floating point value **sval**, and at least one of its four nearest neighbors has intensity less than **sval**, then the corresponding pixel in **dst** is set equal to the floating point value **dval**. Otherwise, **dst** pixel values are left unchanged. **src** should have at least 1 overlap row. **dst** does not need to be the same size as **src**.

CONVOLVE dst, src, opr

Set image **dst** equal to the convolution of image **src** with operator (kernel) **opr**. That is, set each **dst** pixel equal to the sum of $\{\text{src} * \text{opr}\}$, summed over the domain of the operator **opr** with the operator's "center" pixel positioned on the **src** pixel that corresponds to the **dst** pixel. The pixels of image **dst** in the excluded edge region, which is defined by the domain of the operator **opr**, are set to zero. **src** should have enough overlap rows to accommodate operator **opr**; the overlap rows must be present and have valid values. Correct values can be put into the overlap rows with the **OVERLAP** command. The number of overlap rows should be at least as large as the larger of the absolute value of **jmax** or the absolute value of **jmin** for operator **opr**. **opr** must be greater than 0. **dst** and **src** should be different images. The image **dst** does not need to be the same size as the image **src**, but the two images do need to be distributed among the daisies in a manner compatible with the **RESAMPLE** algorithms. If there is question about compatibility, it may be helpful to have extra overlap rows for image **src**.

Note that this is not strictly a proper convolution calculation, which would require

reversing the signs of the two indexes in the operator.

COPDEF dst, src[, novl]

Define space for previously undefined image **dst** so that it is compatible with the previously defined image **src**. That is, the total sizes are the same for the two images, and their primary rows are distributed the same way among the several daisies, so that the two images are compatible for 2- or 3-image operations. The new image **dst** will have **novl** overlap rows regardless of how many overlap rows the old image **src** had. If a value is not given for **novl**, image **dst** will be created with the same number of overlap rows as image **src**. **COPDEF** does not set any pixel values. **COPDEF** sets \$A and \$B to 0.0 and 1.0 for **dst**.

COPY dst, src

Copy image **src** to image **dst**. Both images must be already defined and of the same size.

COPYEDGES dst, src, opr

Copy the values of the pixels at the edges of image **src** to the corresponding pixels of image **dst**. The edge pixels are specified by the domain of the operator **opr**. That is, if a certain pixel of image **src** can be used as the center pixel of the operator **opr** without the domain of the operator **opr** extending off the image **src**, then that certain pixel is NOT an edge pixel and it will not be copied. This same definition of edge regions in terms of an operator is used in many of the operations described later in this report. **opr** should be greater than 0 for this operation, since it would not make sense to copy an edge region which contains no pixels. Images **dst** and **src** should be the same size.

DEFIMG img, col, row, novl

Reserve memory in each daisy to hold part of image **img**. **novl** is the number of overlap rows. The full image comprises **col** columns by **row** rows. If either **col** or **row** is 0, the image **img** is un-defined; its memory space is deallocated and any image data is lost.

You do not have control over which part of the image is held by which daisy. Two images of the same size may be distributed differently among the several daisies, if the two images do not have the same number of overlap rows or if they were created by different processes; such image pairs are not compatible for most multiple-image operations. If two (or more) images are defined by this **DEFIMG** command with the same values of all 3 parameters **col**, **row**, and **novl**, they will be compatible for two-image operations.

DEFIMG does not set pixel values. **DEFIMG** sets the scale factors A and B to 0.0 and 1.0.

DEFKERN opr, imin, imax, jmin, jmax

Reserve memory in each daisy to hold all of the convolution operator (or kernel) **opr**. The parameters **imin**, **imax**, **jmin**, **jmax** define the domain of this operator, in the horizontal and the vertical directions respectively, relative to the "center" pixel. The "center" pixel does not need to be in the center, or even in the two-dimensional domain specified by **imin**,

imax, jmin, jmax. The "center" pixel is merely the pixel with coordinates $i=0$ and $j=0$, which is often, but not necessarily, in the center of the domain. **opr** must be greater than 0. Operator 0 is always automatically defined with **imin, imax, jmin, jmax** all equal to zero; this is useful as a dummy operator with a one-pixel domain. Using operator 0 to specify an excluded edge region implies that none of the image is excluded.

DIV dst, src1, src2

Set image **dst** = image **src1** / image **src2**.

DIVCON dst, src, const

Set image **dst** = image **src** / **const**.

ECHO (no parameters except a remark)

Echo this input line back to the screen if the echo is on.

ECHO_ON (no parameters)

Turn on the echo.

ECHO_OFF (no parameters)

Turn off the echo.

ECHOTIME (no parameters except a remark)

Echo the time and this input line to the screen, regardless of the setting of the echo parameter. This is useful for timing studies and diagnostics.

EDJ02 bright, angle, src, skr, opr, length, width, avg, g

Find edges. This EDJ02 uses XGRAD followed by LIN02 and YGRAD followed by LIN02 to find sharp gradients in the image **src**. Image **bright** is set to the magnitude of the gradient, and image **angle** is set to the tangent of the angle between the gradient direction and the x-axis. The **opr, length, width, and avg** parameters are used in LIN02. **g** is a gradient threshold; gradient amplitudes less than **g** are set to zero and ignored. **skr** is the number of an image that can be used for scratch space. It does not matter whether image **skr** is previously defined; this operation will define, use, and destroy image **skr**. This EDJ02 operation requires one overlap row for image **src**. All four images should be different, all of the same size. **bright** pixels in the excluded edge region defined by the domain of **opr** are set to zero.

END (no parameters)

Stop the feature calculation process for the current scene, and also indicate the end of the feature calculation script in the .fc file. The END differs from the STOP command in that a BRANCH or JUMP command can cause the program to skip over a STOP command and execute later commands, but the program will never skip over an END command and will never read any of the .fc file after the first END command. Each script in a .fc file should contain at least one STOP or END command, and it might contain more than one

STOP command.

EXTRACT *dst, src, row, col*

Copy all of image *dst* from the region of image *src* with top left corner at row *row*, column *col*. If the requested region does not lie entirely within image *src*, part of the destination image *dst* is left unchanged. *dst* and *src* should be different images. This operation does set the overlap rows in *dst*.

FEATFIL (no parameters)

FOR PROGRAM F ONLY. Read a file name from the scene list file and enter that file name into the internal list of feature image file names, without actually reading or writing any image file. The "+" and "." file name constructs, used in the **READSCENE** command, also apply here. This command is useful in the F program when the feature images are the same size as the scene images, in which case the program can use the same file for both feature and scene masks and does not need to write a separate file for the feature mask. Note that the feature mask file name should always be the first name in the list of feature image names.

FEAT* *src1, src2*

FOR PROGRAM F ONLY. The program F maintains an internal list of feature images that have been defined, and this list is used during the calculation of the sums in F. Feature images are defined by a **WRITEFEAT**, a **FEATFIL**, or a **FEAT*** command. They are automatically assigned sequence numbers as they are defined. The first feature image, number 0, must always be the mask. The first "real" feature image is number 1. This operation **FEAT*** puts a special code into the list of defined feature images. The code directs the sums calculation subroutine to use an additional feature image that is not actually stored in a file; this additional feature image is the product (pixel-by-pixel multiplication) of the two previously defined feature images numbered *src1* and *src2*. (*src1* and *src2* may be the same.) This allows the use of a feature image that is the product of two other feature images, without taking the time to write and read a file for this image. This **FEAT*** command should not be used until both of the feature images *src1* and *src2* are defined.

GRADCON *dstb, dsta, srcb, srca*

Concentrate (or sharpen) the peaks in a vector field, in the direction of the vector. This is intended to concentrate a gradient vector field, such as is obtained from operation **GRADT** or from the combination of **XGRAD**, **YGRAD**, and **XY2RT**. Image *srcb* contains the source vector magnitude, image *srca* contains the tangent of the angle between the vector and the x axis, and images *dstb* and *dsta* will contain the corresponding quantities for the concentrated vector field. **GRADCON** assumes that the single rows and columns of pixels at the edges of the source images are all zero, and it sets these edge pixels to zero in *dstb*. This operation uses one overlap row for each of the source images. All the images should be of the same size. The destination images need not be distinct from the source images or from each other. (It does not make sense for the two source images to be the same.) If the two destination images *dstb* and *dsta* are the same, the destination image will contain the

concentrated vector magnitudes. This operation can be repeated to obtain more sharpening, but the OVERLAP operation should be done before each GRADCON operation. This operation is not perfect, but is quite good. There is sometimes some concentration in the wrong direction, which gets worse with repeated application of GRADCON.

GRADT *dstr*, *dstt*, *src*

Set image *dstr* equal to the magnitude of the gradient, and image *dstt* equal to the tangent of the angle between the gradient angle and the x (horizontal) axis, for the gradient of the intensity in image *src*. This operation uses one overlap row for *src*. The destination images need not be different. If the two destination images *dstr* and *dstt* are the same, the destination image will contain the gradient magnitudes and the gradient direction values will not be written to any image. The top and bottom rows, and the left and right columns, of *dstr* are set to zero. Both *dstr* and *dstt* should be different from *src*, and all three images should be of the same size.

HIST2 *src*, *xcl*, *nbin*, *vmin*, *vmax*

Calculate the histogram of intensity values for image *src*. Operator *xcl* defines an excluded edge region; *xcl* may be 0. The histogram will have *nbin* bins representing the pixel values from *vmin* to *vmax*, plus two more bins, for values less than *vmin* (in bin 0) and for values greater than or equal to *vmax* (in bin *nbin*+1). This operation sets the user-accessible variables \$N = number of pixels in the histogram range, \$AVG = average value of those pixels, and \$SIG = standard deviation of those pixel values. These statistics are calculated from the histogram, not directly from the image.

IF *v1*, *op*, *v2*, *command*

Evaluate the logical statement *v1 op v2* and, if the statement is true, execute *command*. *v1* and *v2* are scalars, either constants or user-defined scalar variables. *op* is any of the 6 relational operators "=" (equals), "!=" (does not equal), "<" (is less than), ">" (is greater than), "<=" (is less than or equal to), and ">=" (is greater than or equal to). *command* may be any of the valid commands listed in this appendix. The entire IF command, including *command*, must be on a single line in the .fc file.

INSERT *dst*, *row*, *col*, *src*

This command copies image *src* into image *dst*, with the top left pixel of image *src* going into *dst* pixel (*row*, *col*). This operation copies only that part of *src* that will fit into the specified area of *dst*. *src* and *dst* should be different images. This operation does set the overlap rows in *dst*.

JUMP *label*

Jump to the line numbered *label* in the .fc file. *label* is normally the parameter in a LABEL statement somewhere in the .fc file, in which case this JUMP command causes a jump to the line containing that LABEL command. *label* can also be any constant or user-defined variable whose value is a line number.

LABEL label

Set the value of the user-defined variable **label** equal to the line number of the line containing this LABEL command. If **label** is not already defined, this LABEL command will define it as a type I32 variable. The intent is that the variable **label** will be used with the JUMP command and for no other purpose, but **label** is actually simply another user-defined type I32 variable and it can be manipulated just as any other user-defined variable. Of course, if the value of **label** is changed after it is set by the LABEL command, later JUMP **label** commands may produce undesired results.

LIN01 bright, angle, src, opr, length, width

Find lines in image **src**. Set each pixel in image **bright** equal to the brightness of the line (if any) passing through the corresponding pixel in image **src**, and set the corresponding pixel in image **angle** equal to the tangent of the angle between the line and the x (horizontal) axis. LIN01 will not find lines with negative brightness. **bright**, **angle**, and **src** should all be different images, all the same size. Operator **opr** contains weights for the local region which is analyzed for the presence of a line. **length** is the minimum acceptable line length parameter, and **width** is the maximum acceptable line width parameter. **length** and **width** are measured in pixels, but they are floating point values and fractional parts are meaningful. This operation uses overlap rows for image **src**; it does NOT set any overlap row values in either of the destination images **bright** or **angle**. **bright** pixels in the excluded edge region defined by the domain of **opr** are set to zero.

The LIN01 algorithm is designed to find bright lines on a zero-intensity background, with no negative pixel values. One procedure that is usually reasonable is to subtract a smoothed version of the source image from the raw source image, and then use the MAXCON operation to remove negative pixel values, before using LIN01. Dark lines can, of course, be found by negating an image before starting so that the dark lines appear bright. This algorithm treats the **src*opr** intensity versus position data as a bivariate probability density function, finds the principal axes, and compares the standard deviations in the principal directions with $\text{length}/\sqrt{12}$ and $\text{width}/\sqrt{12}$ to determine whether the distribution is "long" and "narrow" enough to be construed as a line. (For a line of uniform intensity, the standard deviations of the distribution are equal to $\text{length}/\sqrt{12}$ and $\text{width}/\sqrt{12}$.) In other words, this algorithm looks at the peak in the **src*opr** values, regarded as a function of the two position coordinates x and y, and checks to see whether this peak is long and narrow enough to be considered a line. Lines that do not pass through the "central" pixel of the local region defined by **opr** are rejected.

LIN02 bright, angle, src, opr, length, width, avg

This is a line-finding operation like LIN01, except that LIN02 rejects lines if the absolute value of the average intensity in the local region is less than **avg**. LIN02 can find lines with negative brightness.

MAX dst, src1, src2

Set each image **dst** pixel equal to the maximum of the corresponding image **src1** pixel or the corresponding image **src2** pixel.

MAXCON dst, src, const

Set each image **dst** pixel equal to the maximum of the corresponding image **src** pixel or the floating point value **const**.

MEDIAN dst, src, opr[, Nlo[, Nhi]]

Do an order sort filter, in which the output (filtered) value is neither the largest nor the smallest of the values in the local region of the source image. This is a non-linear filter that removes local minima with domains of **Nlo** or fewer pixels, and local maxima with domains of **Nhi** or fewer pixels. For each pixel in image **dst**, **MEDIAN** finds the corresponding pixel in image **src** and aligns thereon the center pixel of operator **opr**. For each non-zero element of **opr**, the corresponding **src** pixel is put into a list of pixel values. The list is sorted according to value. The **dst** pixel value is set equal to the corresponding **src** pixel value, unless this value is smaller than the $(Nlo + 1)$ th smallest value or larger than the $(Nhi + 1)$ th largest value in the list, in which case the limiting list value is used for the **dst** pixel value. Thus, if **Nlo** is 1 and **Nhi** is 2, for example, the **dst** pixel value cannot be the smallest or the largest or the second largest value in the operator domain of **src** pixels. The operator values are not used except to specify which source pixels are in the local neighborhood, which is that part of the operator domain for which the operator values are not zero. **src** should have enough overlap rows to accommodate **opr**. **Nlo** and **Nhi** should normally be greater than 0, and less than half of the number of non-zero operator elements. If either **Nlo** or **Nhi** is -1, that -1 value is replaced by half (integer division by 2) of the number of non-zero elements in the operator. If **Nlo** is missing from the command, -1 is used for the missing value. If **Nhi** is missing from the command line, the value of **Nlo** is used for the missing value. If the operator has an odd number of non-zero elements, and **N** is half of that number (integer division by 2), this **MEDIAN** operation is a standard median filter. Pixels in the excluded edge region of **dst** are set to zero. **dst** and **src** should be different images, and they may be different sizes.

MED1X dst, src, N[, xcl]

Set image **dst** equal to image **src** median filtered with a window of **N** pixels in the x (horizontal) direction by 1 pixel in the y (vertical) direction. The domain of the operator **xcl** defines an excluded edge region, in which **dst** pixel values are left unchanged and **src** pixels are not used. If **xcl** is not given, 0 is used for its value. **dst** and **src** should be different images of the same size. **N** should be an odd integer.

MED1Y dst, src, N[, xcl]

Same as **MED1X**, except this filter is in the y direction. This y-direction filter uses $(N-1)/2$ overlap rows in **src**.

MIN dst, src1, src2

Set each image **dst** pixel equal to the minimum of the corresponding image **src1** pixel or the corresponding image **src2** pixel.

MINCON dst, src, const

Set each image **dst** pixel equal to the minimum of the corresponding image **src** pixel or the floating point value **const**.

MODMSK new, old, res, xcl, region, Q

Set image **new** equal to a mask obtained by modifying the mask in image **old**. Image **res** is a result image from a previous program E calculation. If **region** is 2, then any pixel with value 2 in **old** is changed to 0 in **new** if the **res** pixel value is less than **Q**. If **region** is 1, then any pixel with value 1 in **old** is changed to 0 in **new** if the **res** pixel value is greater than **Q**. If **region** is 3, both operations are done. Other pixels are simply copied from **old** to **new**. Thus, the designated target (2) and background (1) regions in the mask are shrunk so that they do not extend beyond the target and background regions indicated in the result image **res**. This hopefully makes the mask more efficient without damaging its intended target and background designations. Operator **xcl** defines an excluded edge region in which **new** pixels are left unchanged. All three images should be the same size, and **new** may be the same image as either **old** or **res**.

MOMUV src, opr, dst1[, dstu[, dstv[, dstuu[, dstvv[, dsta]]]]]

MOMUV treats the intensity values in the local region as if they were an un-normalized probability density function (PDF), and calculates the second and lower moments of the PDF. MOMUV does a coordinate rotation to maximize the second moment about zero (not about the mean) in the U direction. If the PDF indicates no preferred direction, the U axis is along the x axis (horizontal, positive to the right). Image **src** is the source image, and kernel **opr** defines the local region and the weights for the pixels in the local region. Images **dst1**, **dstu**, **dstv**, **dstuu**, and **dstvv** are destination images for the weighted average intensity, the moments U, V, UU, and VV (the UV moment is always 0), and the tangent of the angle from the x axis to the U axis. If the value 0 is used for any of the destination image numbers, the corresponding quantity is not written to any image. If an image number is not supplied for **dstu** ..., the value 0 is used for the missing image numbers. The x coordinate is positive to the right, and the y coordinate is positive downward, and the origin is at the center pixel of the local region. **src** must have enough overlap rows to accommodate **opr**. The images must all be the same size. **src** should normally be different from all the destination images. The excluded edge pixels in each **dst** are set to zero.

MUL dst, src1, src2

Set image **dst** = image **src1** * image **src2**.

MULCON dst, src, const

Set image **dst** = image **src** * **const**.

NLIN01 dst, bright, angle, opr

Count how many line extensions pass through each pixel. This operation is similar to XLIN01. But, whereas XLIN01 yeilds the sum of the intensities of all the line segments whose extensions would pass through the central pixel, NLIN01 is an attempt to count the

number of line segments whose extensions would pass through the central pixel, independent of the line segment intensities. NLIN01 does not work very well. The weights in the operator **opr** should sum to 1.0 along any one ray from the center pixel.

NTRP00 dst, src, opr

NTRP00 does an interpolation, replacing pixels that have value 0.0 in the source image **src** with new values in the destination image **dst**. The new value is the weighted average of all the non-zero-value pixels in the local region, with the weights contained in the kernel **opr**. **src** should include enough overlap rows to accommodate **opr**. **dst** and **src** should be the same size. **dst** should usually be different from **src**.

NTRP01 dst, src, opr

NTRP01 does an interpolation, replacing pixels that have value 0.0 in the source image **src** with new values in the destination image **dst**. The new value is determined by a weighted least squares fit with a linear function of position to all the non-zero-value pixels in the local region, with the weights contained in the kernel **opr**. **src** should include enough overlap rows to accommodate **opr**. **dst** and **src** should be the same size. **dst** should usually be different from **src**.

OVERLAP img

Set currently correct values in the overlap rows of image **img**. That is, each daisy obtains (from other daisies) the correct, current values of the pixels in its own overlap rows. This should be done before operations like CONVOLVE (convolution calculation) or MED1Y (median filter in the y direction) which use the overlap rows. The overlap rows are automatically assigned the correct values when an image is read from a file (commands READIMAGE and READSCENE). The overlap rows are NOT assigned the correct values for most other operations.

PRINTS [src ...]

Print the values of any scalar variables listed as parameters. If no parameters are listed, all scalar variables are printed.

QUADUV src, opr, dst1[, dstu[, dstv[, dstuu[, dstvv[, dsta]]]]]

Like QUADXY, QUADUV fits the local region with a quadratic function. QUADUV then does a coordinate rotation from the x-y to the u-v coordinates, with the u-axis chosen in the direction that maximizes the second derivative with respect to u. Image **src** is the source image, and kernel **opr** contains the weights for the pixels in the local region. Images **dst1**, **dstu**, **dstv**, **dstuu**, and **dstvv** are destination images for the coefficients of the 1, u, v, uu, and vv terms in the fitted polynomial. (The uv term is always zero.) Image **dsta** is the destination for the tangent of the angle between the x axis and the u axis. If the value 0 is used for any of the destination image numbers, the corresponding quantity is not written to any image.

QUADXY src, opr, dst1[, dstx[, dsty[, dstxx[, dstyy[, dstxy]]]]]

QUADXY does a weighted least squares fit of a quadratic function of position to the pixels in a local region of the source image, and writes the 6 polynomial coefficients to the 6 destination images **dst1**, **dstx**, **dsty**, **dstxx**, **dstyy**, and **dstxy**. If the image number supplied for any destination image is 0, the corresponding quantity is not written to any image. If an image number is not supplied for **dstx** ..., the value 0 is used for the missing image numbers. The x coordinate is positive to the right, and the y coordinate is positive downward, and the origin is at the center pixel of the local region. Image **src** is the source image. Kernel **opr** defines the local region and contains the weights. **src** must have enough overlap rows to accommodate **opr**. The images must all be the same size. **src** should normally be different from all the destination images. The excluded edge pixels in each **dst** are set to zero.

READIMAGE img[, header[, row0[, col0[, ncols[, bpp]]]]] / file name

Read image **img** from file **file name**. A value of 0 for **header** means that the image file does not have a header; a value of 1 means that the image file has a Data Translation header; a value of 2 implies a Perceptron 11-byte header. The memory image **img** may be smaller than the file image in **file name**; the memory image is taken from that region of the file image with top left corner at row **row0** and column **col0**. **ncols** is the number of columns in the file image, unless **ncols** is 0 in which case the number of columns in the file image is assumed to be the same as the number of columns in the memory image. Thus, for the common case in which the file image is the same size as the memory image, **row0** and **col0** and **ncols** can all be set to 0. **bpp** is the number of bytes per pixel. Acceptable values are 1; 2, implying the less significant byte of each pair is first in the file; and -2, implying two bytes per pixel with the more significant byte first. **img** must be greater than 0. The file name must be entered on a separate line, the line following the READIMAGE command and its other parameters, in the feature calculation command file. Unlike many operations, READIMAGE does set the values of the pixels in the overlap rows of image **img**.

If the first line of this command includes fewer than 6 parameters after the READIMAGE keyword, the last parameters in the list are assumed to be missing and the previously specified values for the missing parameters are used here. Initial values are **header** = 1, **row0** = 0, **col0** = 0, **ncols** = 0, and **bpp** = 1. The **img** parameter must always be supplied with this command.

The image may be already defined (using DEFIMG, COPDEF, RESAMPLE, or a previous READIMAGE or READSCENE), in which case it is assumed that the memory image size, the file image size, and the command parameters are all compatible. If image **img** is not already defined, and **header** is 1 or 2, the image **img** will automatically be defined to match the image in the file, with 0 overlap rows. If **row0** is 0 and **header** is 1, either explicitly or by default, the program will attempt to read the values of the scale factors A and B from the header in the image file, and set the image table A and B values and the pre-defined scalar \$A and \$B values accordingly. Otherwise, these will be set to 0.0 and 1.0.

READKERNEL opr / file name

Read a local convolution operator (or kernel) from file **file name** into operator buffer number **opr**. The program assumes that the operator buffer has already been defined (operation DEFKERN) and that the buffer and the file are compatible. **opr** must be greater than 0. The file is assumed to contain ASCII characters, with the several values separated by spaces, commas, or end-of-line characters.

READSCENE img[, header[, row0[, col0[, ncols[, bpp]]]]]

Read an image from the file whose name is in the scene image list file (.sl). This READSCENE is like READIMAGE, except that for READSCENE the file name is not given in the feature command file (.fc) but is given instead in the scene list file (.sl).

The program remembers previously specified names of scene files. If, instead of a file name, the entry in the scene list file (.sl) is a "+" followed by an integer value *n*, then the previously used file name for the *n*-th scene image is incremented and used as the present file name. If a "+" is given without an integer value, the value of *n* is assumed to be the same as the number of the present scene image. For example, assume that this is the third call to READSCENE in the feature command file (we are getting scene image 3), and that we are processing the fifth scene (this is the fifth use of the feature command file in this execution of the program). This call will use the third scene image file name in the fifth block of file names in the scene list file (.sl). If the file name is "+", the file name used for the image 3 (the current scene image number) in scene 4 (the previous scene number) will be incremented and then used here. If the file name is "+2", the file name to be incremented and used here will be the one for image 2 of scene 5, which is the last image 2 file name specified. If the file name is "+4", the file name to be incremented and used here will be the one for image 4 of scene 4, since the file for image 4 of scene 5 has not yet been specified.

A "." has an effect similar to a "+", except that the old file name is not incremented before being used.

Each time the READSCENE command is used after the scene list file (.sl) is ended, the effect is the same as if "+" were read from the scene list file.

"Incrementing a file name" means that we treat the base file name, exclusive of the path and extension, as if it were a kind of string of digits representing a number, and we increase it by one. For example, C:FILE15.IMG increments to C:FILE16.IMG; NAME29 to NAME30; FILE9 to FILE0; TESTA to TESTB; F7Z to F8A; XZZ to YAA; TT99 to TU00; etc. The path and extension are never changed. Alphabetic characters always increment to other alphabetic characters, and numerals to other numerals.

The scene list file (.sl) contains a block of image file names for each scene. For program E, the first file name in each block is the name of the result file to be written, and the following file names in the block are real scene image file names numbered 1, 2, ... For program F, there is no result image written, and the block of file names in the scene list file does not start with a result image file name. Instead, for program F, the file names in a block are all treated as scene image file names, and they are numbered 0, 1, 2, ... It is often convenient to use the mask image file name for the first scene image file name, so that the mask will be "scene image" number 0 and the "real" scene images will be numbered 1, 2, ...

Note that the first feature file written for each scene by program F must be the mask feature file, so it is handy if the first scene image file read is the scene mask image file.

READY (no parameters)

Query all slaves, including the SCSI and video interfaces, and wait for each to respond. This **READY** command causes the system to wait until all nodes have finished their tasks, which is useful for timing measurements and for debugging.

Usually, the fact that a slave responds to this or any other input indicates that the slave has finished previous tasks. However, if the slave is doing an operation that requires communication with other nodes, this **READY** query may cause an error condition in the slave program. Correctly written master node subroutines will prevent this query or any other communication from being sent to a slave while the slave is in such a vulnerable situation; the master node subroutine should not allow the input of any user command until all the slave nodes are no longer vulnerable.

REM (no parameters except a remark)
no action.

REPHI *dst, src, const, rep*

Set image *dst* pixel equal to the corresponding image *src* if that *src* pixel value is no greater than *const*; otherwise, set the *dst* pixel value to the floating point value *rep*.

REPLO *dst, src, const, rep*

Set image *dst* pixel equal to the corresponding image *src* if that *src* pixel value is at least as great as *const*; otherwise, set the *dst* pixel value to the floating point value *rep*.

RESAMPLE *dst, src, xcl, ncol, nrow, novl*

Resample image *src* and put the result into image *dst*. The operator *xcl* defines an excluded edge region of image *src*; *xcl* may be 0. There is no excluded edge in *dst*. If image *dst* is not yet defined, it will be defined with *ncol* columns, *nrow* rows, and *novl* overlap rows, and its scale factors A and B will be set to 0.0 and 1.0. Note that images defined by this **RESAMPLE** operation may be distributed among the daisies differently than images of the same size defined by other processes. If image *dst* was previously defined, it is assumed to have been defined with the correct size and with the correct parts allocated to each daisy, and the values supplied for *ncol*, *nrow*, and *novl* are ignored. The values of the overlap row pixels in *dst* are NOT set by this **RESAMPLE** operation. The resampling is done by the simplest nearest-pixel method, with no interpolation or smoothing. If image *src* has fewer rows than image *dst* and image *src* has overlap rows, the overlap rows will probably be used in the **RESAMPLE** procedure, so their values (if not already current) should be updated with the **OVERLAP** command before this **RESAMPLE** command is used. *src* overlap rows may also be used if the *src* and *dst* images are not distributed exactly correctly among the slave nodes, as might happen if one of the images is defined by a process other than a **RESAMPLE** of the other image.

RIJCON dst, src, opr

This is an imperfect operation that tries to concentrate ridges. A weighted least squares fit is used to fit a quadratic polynomial to the local region in the source image **src**, with kernel **opr** containing the weights and specifying the local region. If the fitted polynomial describes a ridge, the ridge is smoothed along its length and concentrated in the direction perpendicular to its length, in the destination image **dst**. **dst** should be different from **src**, and they should be the same size. **src** should include enough overlap rows to accommodate **opr**. The excluded edge pixels in **dst** are set to zero.

SABS dst, src

Set scalar variable **dst** = absolute value of **src**.

SADD dst, src1, src2

Set scalar variable **dst** = **src1** + **src2**.

SCALE dst, src[, xcl]

Image **dst** is set equal to image **src** multiplied by a scale factor chosen to make the largest value in image **dst** equal to 254. The edge pixels, defined by operator **xcl**, are excluded; their values in **src** are not considered in determining the scale factor, and their values in **dst** are left unchanged. The image **dst** coefficient A is set to 0.0 and B is set to the scale factor used in this scaling operation, without regard to the coefficient values for image **src**. **dst** and **src** should be of the same size, and they may be the same image. Usually, **SCALE** should be used immediately before **WRITEFEAT** and other 8-bit image writing operations to be sure that the limited dynamic range of 8 bits per pixel is best utilized. **SCALE** is also useful before screen display operations.

SDEFF32 src1 [src2 ...]

Define 32-bit floating point scalar variables with names **src1**, **src2**, ..., similar to **SDEFI32**.

SDEFI32 src1 [src2 ...]

Define 32-bit integer scalar variables with names **src1**, **src2**, ... Variable names are limited to 30 characters. You must not try to re-define an already defined variable. This operation does not assign values to the newly-defined variables.

SDIV dst, src1, src2

Set scalar variable **dst** = **src1** / **src2**.

SEGLAB dst, src

SEGLAB assigns unique labels to image segments. **SEGLAB** assumes that image **src** has patches of marked pixels, with values greater than 0, separated by regions of background pixels, with values less than or equal to 0. For each marked **src** pixel, **SEGLAB** sets the corresponding destination image **dst** pixel to some value that comprises a label for the segment. Each contiguous segment gets a unique label. **dst** pixel values are not changed if

the corresponding **src** pixels are not marked as segments. Normally, all **dst** pixels should be set to 0 before starting a segmentation procedure that uses this subroutine. **dst** pixel values that are not 0 on entering this subroutine are construed as indicating a pixel that is already labeled. A non-zero value for a **dst** pixel and a greater-than-zero value for the corresponding **src** pixel is an inconsistency, indicating overlapping patches (segments). When this occurs, the old labels will not be changed, and the non-overlapping parts of the new overlapping patches will get a new label. Successive uses of the same destination image with different source image will result in unique labels for all the patches (segments) in the several source images as long as those patches do not overlap. **src** and **dst** must be different images, they must be of the same size, and both **dst** and **src** must have at least one overlap row.

SEQ **dst, **src****

Set scalar variable **dst** = **src**.

SET_AB **img, **A**, **B****

Set the **A** and **B** coefficients for image **img** to floating point values **A** and **B**. ([This image] = **A** + **B***[original image].) These coefficients are initially set to **A**=0.0, **B**=1.0. This operation does NOT rescale the image; it merely sets the values of the stored coefficients.

SETPIX **img, **row**, **col**, **val****

Set the pixel in image **img**, row **row**, column **col**, to the floating point value **val**. This operation sets pixels in both the primary and the overlap rows. Note that images are stored in memory as floating point data.

SMAX **dst, **src1**, **src2****

Set scalar variable **dst** = maximum of **src1** or **src2**.

SMIN **dst, **src1**, **src2****

Set scalar variable **dst** = minimum of **src1** or **src2**.

SMTHX **dst, **src**, **sigma**, **N**[, **xcl**]**

Set image **dst** equal to image **src** smoothed in the **x** (horizontal) direction. The smoothing operation comprises **N** repetitions of a two-pass (one pass increasing **x**, one pass decreasing **x**) exponential smoothing filter, with the exponential filter width chosen to make the half-width (the standard deviation) of the smoothing function of the total operation equal to the floating point value **sigma**. Larger **N** values make the overall smoothing function look more Gaussian. The domain of the operator **xcl** specifies an excluded edge region, in which the **dst** pixel values are not changed. (The operator **xcl** is not used for anything except specifying this excluded edge region.) **dst** should be the same size as **src**, and they may be the same image.

SMTHY **dst, **src**, **sigma**, **N**[, **xcl**]**

Same as **SMTHX**, but this smoothing is in the **y** (vertical) direction. (Overlap rows

are not needed.)

SMUL dst, src1, src2

Set scalar variable **dst** = **src1** * **src2**.

SQRT dst, src

Set image **dst** = square root of image **src**.

SSQ dst, src, opr

Set image **dst** equal to the local sum of squares of image **src**, with weighting coefficients from operator **opr**. That is, set each image **dst** pixel value equal to the sum over the domain of operator **opr** of {**src*****src*****opr**}. The **dst** pixels in the excluded edge region, defined by the domain of the operator **opr**, are set to zero. **src** should have enough overlap rows to accommodate operator **opr**. **opr** must be greater than 0. **dst** and **src** should be different images. The image **dst** does not need to be the same size as the image **src**, but the two images do need to be distributed among the daisies in a manner compatible with the **RESAMPLE** algorithms.

SSQRT dst, src

Set scalar variable **dst** = square root of **src**.

SSUB dst, src1, src2

Set scalar variable **dst** = **src1** - **src2**.

STATS img[, xcl]

Calculate pixel value statistics for image **img**, excluding the edge region defined by the domain of operator **xcl**. If **xcl** is not given, the value 0 is assumed and no edge region is excluded. This operation sets the user-accessible variables **\$N** = number of pixels, **\$AVG** = average value of pixels, **\$SIG** = standard deviation of pixel values, **\$MAX** = maximum pixel value, **\$MIN** = minimum pixel value, **\$NROW** = number of rows, **\$NCOL** = number of columns, **\$A** = value of A coefficient in tables, **\$B** = value of B coefficient in tables.

STOP (no parameters)

Stop the feature calculation process for the current scene. For program E, go on to the next scene. For program F, accumulate the sums for this scene and then go on to the next scene, optimizing the coefficients after the last scene.

SUB dst, src1, src2

Set image **dst** = image **src1** - image **src2**.

SUBCON dst, src, const

Set image **dst** = image **src** - **const**.

TTG3 *img*[, *row0*, *col0*[, *Q*]] / *label*

Display image ***img*** on the high-resolution RGB monitor via the TTG3 transputer-video interface. The top left corner of the image will appear at ***row0,col0*** in the screen display. Red will be used for pixels with value less than ***Q***, green for pixels with value greater than ***Q***. If a value is not given for ***Q***, the value 0.0 is used. If values are not given for ***row0*** and ***col0***, values 0 and 0 are used. Pixel values should be between 0.0 and 255.0 (see **SCALE**). The character string ***label*** will be printed below the image on the screen. ***label*** should not contain any embedded spaces. If a "." is the first character of the string given as the label, then the last-used file name will be used as the label. The image will remain on the screen, and the program will stop execution, until the operator hits a key on the keyboard.

TTG3RES *img*, *row0*, *col0*, *Q*, *res* / *label*

Display image ***img*** on the high-resolution RGB monitor via the TTG3 transputer-video interface. The top left corner of the image will appear at ***row0,col0*** in the screen display. Red will be used for those pixels with value less than ***Q*** in the result image ***res***, green for pixels with value greater than ***Q*** in the result image ***res***. ***res*** and ***img*** should be the same size. ***img*** pixel values should be between 0.0 and 255.0. The character string ***label*** will be printed below the image on the screen. ***label*** should not contain any embedded spaces. If a "." is the first character of the string given as the label, then the last-used file name will be used as the label. The image will remain on the screen, and the program will stop execution, until the operator hits a key on the keyboard.

UNDERSAMPLE *dst*, *src*, *xcl*, *xfact*, *yfact*, *novl*

Undersample image ***src*** and put the result into image ***dst***. This function is the same as **RESAMPLE**, except that, whereas **RESAMPLE** specifies the size of the new image ***dst*** directly, **UNDERSAMPLE** uses ***xfact*** and ***yfact*** to specify the size of the new image ***dst*** relative to the size of the old image ***src***. The size of the new image ***dst*** will be the size of the old image ***src*** minus the excluded edge regions defined by the domain of the operator ***xcl***, divided by ***xfact*** in the horizontal direction and ***yfact*** in the vertical direction. Images defined by this **UNDERSAMPLE** operation are completely compatible with (are distributed among the slave nodes the same way as) images defined by the **RESAMPLE** operation.

VGA *img*, *row0*, *col0*, *Q* / *label*

Display image ***img*** on the system (console) monitor. The top left corner of the image will appear at ***row0,col0*** in the screen display. Red will be used for pixels with value less than ***Q***, green for pixels with value greater than ***Q***. Pixel values should be between 0.0 and 255.0. The character string ***label*** will be printed below the image on the screen. ***label*** should not contain any embedded spaces. If a "." is the first character of the string given as the label, then the last-used result file name will be used as the label. This requires that Gordon Lassahn's device handler **VGA** and compatible video hardware are installed.

WRITEFEAT *src* / *file name*

FOR PROGRAM F ONLY. Write image ***src*** to file ***file name***, as in operation **WRITEIMAGE**, and enter ***file name*** into an internal list of feature image files. If a string

beginning with "+" is given instead of a file name, the previously used feature image file name is incremented and used here. Feature images are always written with a Data Translation header, in which is embedded the value of the last scale factor B used in scaling the image with the SCALE command.

Feature image files are written with each pixel represented as one 8-bit (1-byte) integer. This means that any pixel value outside the range 0.0 to 255.0 will not be correctly represented. It also means that precision is limited. As a worst case example, if the feature image pixel values range from 0.0032 to 0.4158, they will all be written as 0 and all precision will be lost. You must construct feature images in such a way that there are no negative values. After this is done, you should usually use the SCALE command just before the WRITEFEAT command. This scale command multiplies the image by a constant that allows optimal use of the available precision, and this constant is written to the feature image file and it is used later by the F program to restore the feature image pixels to their original (before scaling) values during the sums calculations.

WRITEIMAGE src[, header] / file name

Write image number **src** to file **file name**. If the value of **header** is 0, no header is written to the file; if the value is 1, a Data Translation header is written. If **header** is not given, a value of 1 is used. Bytes 127 through 152 of the Data Translation header (the first byte of the header is called number 1) will be the scale coefficients A and B in FORTRAN format (1x,e12.6,1x,e12.6).

WRITERESULT src, header, Q

FOR PROGRAM E ONLY. This is the same as command WRITEIMAGE, with two additional features. The value **Q** is stored in the file header (if a header is written). The file name for this result image output file is read from the scene list file (.sl), the first of the set of file names given for the scene, the name that takes the place of the name of the ~~mask~~ image in the scene list file used for the training process. If "+" is given instead of a file name in the scene list file (.sl), then the previously used result image file name is incremented and used here.

XGRAD dst, src

Set image **dst** equal to the x-gradient (derivative with respect to x, the coordinate that increases from left to right) of the intensity in image **src**. The gradient values are calculated as $\text{gradient} = (\text{intensity}[i+1] - \text{intensity}[i-1]) / 2$. The first and last columns (left and right edges) of **dst** are set to 0. **dst** and **src** should be different images, of the same size.

XLIN01 dst, bright, angle, opr

Extend lines found by LIN01. The "central" pixel in the local region of image **dst** is set equal to the weighted sum of the brightnesses of the line segments that are in the local region defined by the operator **opr** and are oriented in the direction so that their extensions would pass through the central pixel. The weights are the values of the pixels in **opr**. **bright** and **angle** are the images containing the brightness and orientation data from LIN01. **dst**, **bright**, and **angle** should all be different images, all of the same size. This operation

uses overlap rows for both images **bright** and **angle**. **dst** pixels in the excluded edge region defined by the domain of **opr** are set to zero.

XY2RT dst, dstt, srcx, srcy[, xcl]

Convert a vector field from x-y representation to r-tangent(angle) representation. Images **srcx** and **srcy** contain the x and y component values. The vector magnitude will be put into image **dst**, and the tangent of the angle between the vector and the x axis will be put into image **dstt**. **xcl** specifies an excluded edge region, in which the **dst** and **dstt** pixel values will not be changed. None of the images need be different from any of the others, although it would normally not be sensible for **srcx** and **srcy** to be the same image. If the two destination images **dst** and **dstt** are the same, the destination image will contain the vector magnitudes and the vector direction values will not be written to any image. All these images should be the same size.

YGRAD dst, src

Set image **dst** equal to the y-gradient (derivative with respect to y, the coordinate that increases from top to bottom of an image) of the intensity in image **src**. The gradient values are calculated as $\text{gradient} = (\text{intensity}[j+1] - \text{intensity}[j-1]) / 2$. The top and bottom rows of **dst** are set to 0. **dst** and **src** should be different images, of the same size. This operation requires one overlap row for image **src**.

ZEROIMAGE img

Set all the pixels in image **img**, including the overlap rows, to zero.

DATE

FILMED

8/26/94

END