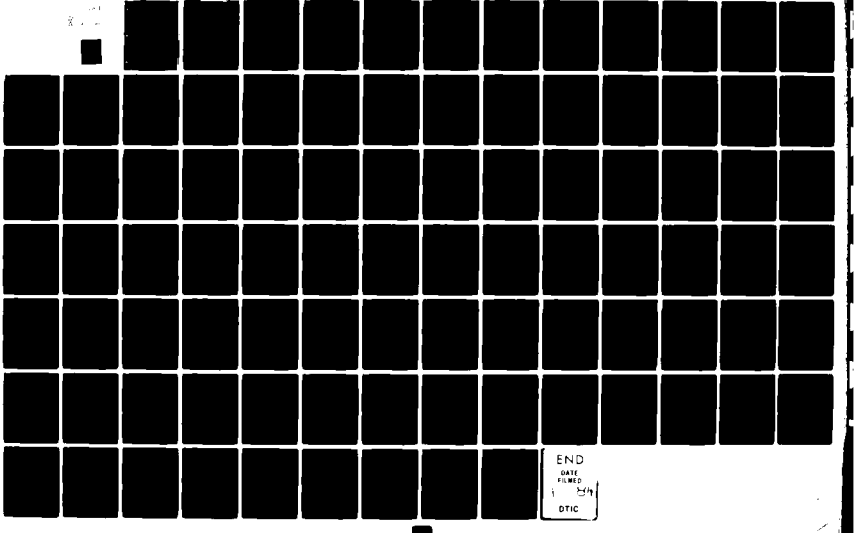
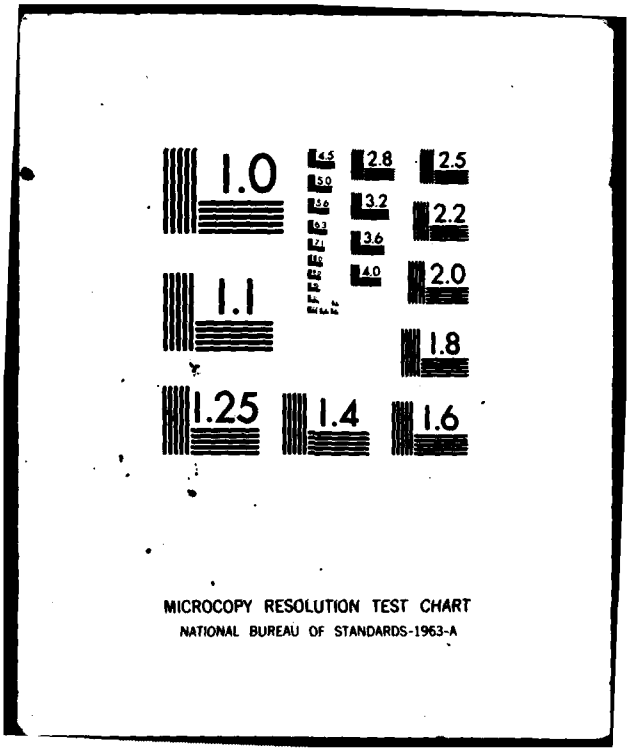
AD-A091 928 GOODYEAR AEROSPACE CORP AKRON OH F/G 9/2
STARAN IMAGE PROCESSING.(U)
OCT 80 R W LOTT, S G STADELMAN, R O FAISS DAAK70-79-C-0221
UNCLASSIFIED GER-16902 ETL-0243 NL



END
DATE FILMED
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

ETL-0243

STARAN IMAGE PROCESSING

R. O. Faiss
R. W. Lott
S. G. Stadelman

FINAL TECHNICAL REPORT
OCTOBER 1980

Approved for Public Release:
Distribution Unlimited

Prepared for

U. S. Army Engineer Topographic Laboratories
Fort Belvoir, Virginia 22060

Prepared by

Goodyear Aerospace Corporation
1210 Massillon Road
Akron, Ohio 44315

Destroy this report when no longer needed.
Do not return it to the originator.

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

The citation in this report of trade names of commercially available products does not constitute official endorsement or approval of the use of such products.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(19) REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (18) ETL-0243	2. GOVT ACCESSION NO. AD-A091 928
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle) (6) STARAN Image Processing.	5. TYPE OF REPORT & PERIOD COVERED Final Sept. 1979 Oct. 1980
6. PERFORMING ORG. REPORT NUMBER (14) GER-16902	
7. AUTHOR(s) (10) R. W. Lott, S. G. Stadelman, R. O. Faiss	8. CONTRACT OR GRANT NUMBER(s) (15) DAAK70-79-C-0221 new
9. PERFORMING ORGANIZATION NAME AND ADDRESS Goodyear Aerospace Corporation 1210 Massillon Road Akron, Ohio 44315	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (12) 91
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Engineer Topographic Laboratories Fort Belvoir, Va. 22060	12. REPORT DATE (11) Oct 1980
	13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (9) Final rept. Sep '79-Oct 80	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Parallel processing, relaxation labeling, edge enhancement

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report discusses an image edge pixel enhancement technique that has been implemented on the joint CDC- 6400 STARAN computer facility at ETL. The algorithm employs the interactive processes typical of relaxation labeling.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

156800 JOB

SUMMARY

The primary objective of this work was the implementation of a label relaxation edge enhancement algorithm for on the STARAN computer. The algorithms were designed to allow for various user defined control parameters so as to be able to evaluate the method of edge enhancement under different conditions. During software execution, STARAN handles the bulk of the computational tasks while the CDC 6400 serves as the I/O and process controller. The CDC 6400 control and I/O software is given access to the STARAN software via calls to CDC resident FORTRAN subroutines. As a result, the general CDC 6400 FORTRAN programmer can incorporate the functional capabilities of the developed STARAN subroutine into his own program.

The algorithm develops likelihood (probability estimates) values of directed edges in an image for 8 compass directions. 3x3 correlation reference patterns are used to develop the initial likelihood values. The likelihood values are then iteratively enhanced by a relaxation process. The likelihood enhancement process involves two basic steps: 1) the best link of likelihood ratio estimates between adjacent neighbor pixel paris is established. In the second step, the best link pair information is used to update the edge likelihood ratio. The CDC:STARAN system can process one iteration in under one minute, thus providing an interactive evaluation capability.

PREFACE

This document is generated under Contract DAAK-79-C-0021 for the U.S. Army Engineer Topographic Laboratories, Fort Belvoir, Virginia, 22060, by Goodyear Aerospace Corporation (GAC), Akron, Ohio, and submitted by GAC as GER-16902. The Contract Officer's Representative was Mrs. Jane Brown.

TABLE OF CONTENTS

INTRODUCTION

This report discusses an image edge pixel enhancement technique that has been implemented in the STARAN computer. The procedure employs the interactive processes typical of relaxation labeling procedure. The software system that implements the procedure is installed on the Army's Engineering Topographic Laboratory, Fort Belvoir, Va. joint CDC 6400/STARAN computer facility. During software execution, STARAN handles the bulk of the computational tasks while the CDC 6400 serves as the I/O and process controller. The CDC 6400 control and I/O software is given access to the STARAN software via calls to CDC resident FORTRAN subroutines. As a result, the general CDC 6400 FORTRAN programmer can incorporate the functional capabilities of the developed STARAN subroutines into his or her own program.

The discussion that follows describes the rationale for the edge enhancement algorithm, discusses the general and specific character of the algorithm, and then discusses the character and use of the software developed to implement the algorithm.

A. TECHNICAL DISCUSSION

1. Background

The manual analysis of imagery, photographic or digital, is both costly and time consuming. In the past, the expense of manual analysis was tolerated because automatic analysis was presumed to be even more costly to accomplish. Continuing advances in the ability to collect imagery data (particularly in direct digital form) and the rapid advance of the computational capability of digital processing hardware is forcing a serious effort to develop automatic analysis techniques and systems.

At present, scene analysis presumes that an image consists of elemental areal sub regions, each of which has its own distinct attributes. It is assumed that once the sub regions are defined by boundary lines and an internal region description vector, the effect of imagery scale, imagery orientation, and imagery intensity can be excluded from the analysis of conglomerates of sub regions. It is presumed that the existence of a pattern, either macroscopic (large count conglomerates) or microscopic (small count conglomerates), can then be automatically ascertained.

This study was restricted to one subset of the problem of automated scene analysis, namely, the subject of automatically detecting region boundaries of a scene. In general, the boundries between two regions can be based on many different attributes such as textural differences, velocity differences (in multi scene analysis) and reflectance (gray scale) values. Reflectance based boundries or "edges" are emphasised in this study.

To assign a label of "edge" or "no-edge" to a pixel. It is common to measure the gradient of the reflectance values in a neighborhood centered about the pixel. In a small neighorhood the assumption that a steep gradient represents a straight line edge is likely to be valid. Moreover, estimates of orientation of the line are likely to reasonable also. Unfortunately, when the neighborhood is small, the number of pixels available to make the gradient estimate is small and so the estimated value has a high variance associated with it. On the other hand, if the neighborhood region were allowed to become larger, estimates of the gradient would tend to become better, except that the uncertainty of the demarcation line shape factor would increase. The interplay of these influences are often limited to 3x3,5x5 and 7x7 neighborhoods because of computational limitations.

Iterative relaxation is one approach to attempt to achieve more reliable edge definitions by using information over larger regions. The basic principle behind label relaxation is that adjacent labels (edge value assignments) should reinforce each other if they are in agreement, counter act each other if they are in disagreement and have no influence on each other if there is a neutral relationship between them.

2. Selected Edge Enhancement Algorithm Discussion

To locate pixels on the edges the gradient of the average reflectance estimates can be used. Provided that the gradient is sufficiently large, the pixel separating the two neighborhood sub regions can be labeled "edge." It may be noted that an image correlation procedure that makes use of a 2-D step reference function provides all the tools needed to generate the neighborhood sub region differences. The orientation of the step edge of the reference function defines the test demarcation line direction, the low side of the step develops the estimate of average reflectance of one neighborhood sub regions, and the high side of the step develops the corresponding estimate for the second neighborhood sub region. The selected correlation process for this study provides the difference of the reflectance estimates directly.

The demarcation step between two regions can range from small to large and so it would be difficult to set an absolute threshold for making the decision "edge" or "not edge" without specific knowledge of the means of the regions being separated. The separate measures of average level estimates are not presently developed during the correlation process and so cannot be used in a normalization scheme. What can be used for normalizing the decision process are appropriately combined estimates of the noise (or jitter) associated with each of the neighborhood pixel regions. By dividing the difference of the neighborhood sub region reflectance estimates by the estimates of "jitter" (standard deviation of reflectance) in the neighborhood, the ratio that represents the likelihood that the sub region reflectance difference is real is obtained. Labeling of "edge" or "no edge" can then occur via a simple threshold test.

It may be noted that in tests to date, the neighborhood of a pixel is a 3x3 region. Since this region must be divided into two sub regions, the average size of the neighborhood sub region is only 4.5 pixels. Necessarily, the smaller sub region could be no larger

than 4 pixels in size. In fact, only a 3 and 6 pixel sub region pattern has been employed to date. When sub regions are this small, poor estimates of the reflectance means of each sub region result. Necessarily, the reflectance difference tends to exhibit a very high standard deviation.

The estimate of reflectance difference uncertainty (or jitter) uses data from the same neighborhood region as was used to establish the difference estimate. This estimate once again suffers because of the smallness of the neighborhood region. Moreover, because the jitter estimates for the present implementation are not made independently for the two regions, the jitter estimate near edges is distorted. As a result, for large steps, the jitter size is highly exaggerated with the result that where a true step occurs, the normalization denominator is abnormally large. As a direct consequence, the ratio of the region reflectance difference to the jitter estimate is substantially diminished. It would be desirable to correct both weaknesses of the present implementation noted above.

3. Method of Initial Likelihood Estimate Computation

The relaxation labeling algorithm develops and then iterates upon likelihood (probability estimates) values that directed edges exist for an image in the compass directions. Appropriate 3x3 correlation reference patterns are used to develop the initial likelihood values for the eight directions as follows:

a_{ij} is an unscaled reference pattern element.

$X_{k\ell}$ is an image pixel.

$\mu_a = \frac{1}{n^2} \sum_i^n \sum_j^n a_{ij}$ is the average reference pattern value.

$$\sigma_a = \frac{1}{n}\sqrt{\sum_i^n \sum_j^n (a_{ij} - \mu_a)^2}$$ is the standard deviation of the reference pattern.

$a'_{ij} = (a_{ij} - \mu_a)/\sigma_a$ are the modified reference pattern values.

The likelihood (probability) of an elemental edge passing through the pixel at $X_{k\ell}$ in direction d is given by

$$P^d_{k\ell} = \frac{\sum_i^n \sum_j^n a'^d_{ij} \cdot X_{r+i,s+j}}{\sigma_{X_{k\ell}}}, \quad r = k - (\tfrac{n+1}{2}), \quad s = \ell - (\tfrac{n+1}{2})$$

where the numerator is the reflectance difference estimate, and the denominator is the jitter estimate. What is meant by "elemental edge" is a small segment of edge at the current pixel position. These elemental edges are connected together during the relaxation process that follows. It may be noted that $P^d_{k\ell}$ is signed because the difference between the reflectance means of the two sub regions can be positive or negative. Because $P^d_{k\ell}$ represents a scaled reflectance difference over a fixed spatial interval, it has the character of a reflectance gradient. In fact, for any given reference pattern, $P^d_{k\ell}$ is the normalized reflectance gradient vector that exists normal to the hypothesized line of demarcation between the pattern's neighborhood sub regions (see Figure 1).

The reference patterns used at a pixel must be input to the FORTRAN subroutine RELAX in a fixed, specific order as shown in Figure 1.

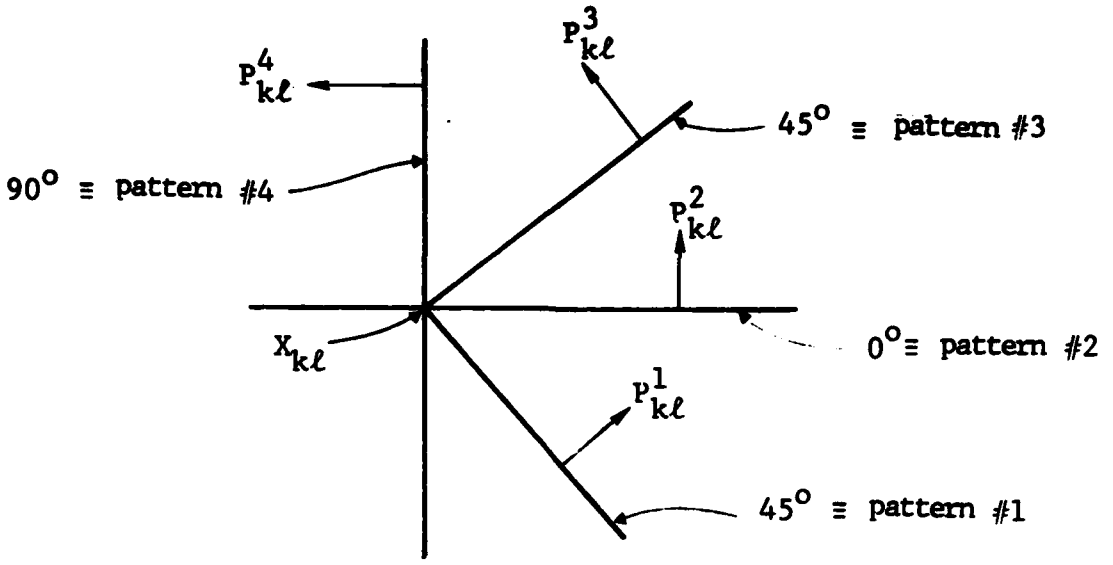
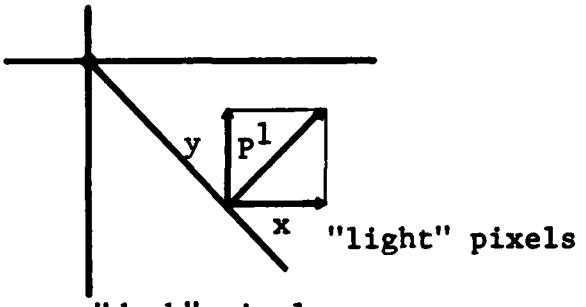


Figure 1. $P^d_{k\ell}$: Reference Pattern: Edge Direction Relation

Thus, reference pattern #1 is used to define subregions that are separated by an elemental edge that lies along the -45° azimuth, etc. As shown in Figure 1, the P^d values represent normalized reflectance gradient vectors that lie normal to subregion demarcation lines. The length of a P^d vector is a measure of strength or the gradient value that an edge exists as one travels across an associated azimuth.

Each P^d is converted to its x,y vector components. The sign of the x,y component values indicates a light-to-dark transition or dark-to-light transition when traveling across an edge as follows using P^1 as an example:

a) x,y are both positive, i.e., a positive correlation with reference pattern #1.

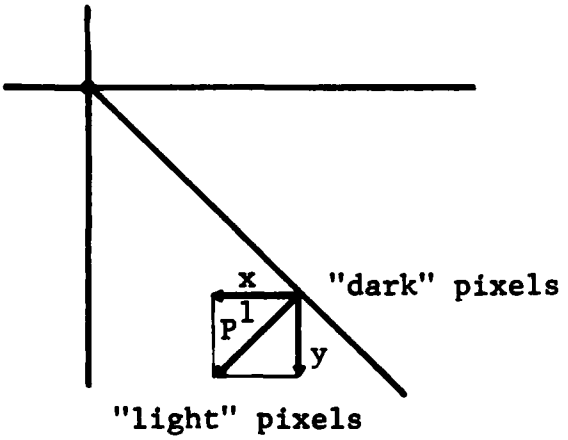


example of
reference pattern #1

1	1	1
-2	1	1
-2	-2	1

Note that the most significant bit of all pixels processed in the STARAN is complemented. Thus, rather than working with unsigned 8-bit pixels ranging from 0 to 255 (black to white), the STARAN deals only with signed 8-bit pixels ranging from -128 to +127 (black to white). Therefore, correlation of $\pm$ pixel values with $\pm$ reference pattern values is straightforward.

b) x,y are both negative, i.e., a negative correlation with reference pattern #1.



example of
reference pattern #1

1	1	1
-2	1	1
-2	-2	1

4. Method of Relaxation

Once a likelihood ratio for a pixel has been established, it could be labelled "edge" or "not edge." Yet it can be useful to defer labeling provided additional processing decreases the variance associated with the likelihood ratio. The process that improves the likelihood ratio hangs on the assumption that the elemental edges associated with a pixel must show continuity with the elemental

edges of neighbor pixels. Furthermore, it is assumed that a minimum curvature edge line between pixels is more likely than an edge exhibiting curvature. (The assumption is observed to be realistic for edges in scenes except where corners are formed.)

The likelihood enhancement process, which is made iterative, involves two basic steps. In the first step, the best link of likelihood ratio estimates between adjacent neighbor pixel pairs is established. In the second step, the best link pair information is used to update the edge likelihood (probability) ratio.

The neighbor's of reference pixel $X^R_{k\ell}$ that take part in the iterative enhancement process are shown in Figure 2.

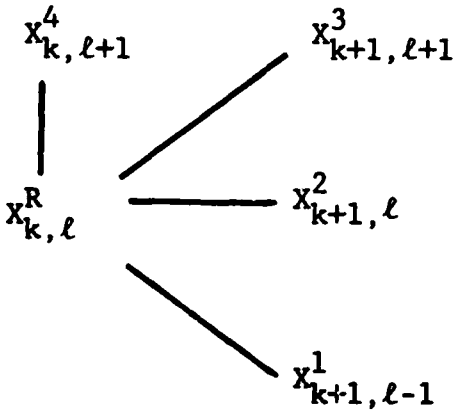


Figure 2. Neighbor's of the Reference Pixel

The superscripts on the neighbor positions identify the four link directions, d= 1,2,3,4. At the conclusion of the initial likelihood estimate computation, the original pixel values have been replaced by 4-tuples: $X^R_{k,\ell} \equiv \{(x^R_1,y^R_1), (x^R_2,y^R_2), (x^R_3,y^R_3), (x^R_4,y^R_4)\}$, where (x^R_d,y^R_d) are the vector components of the likelihood that an edge exists when traveling across the azimuth of direction d.

The best link equation, M_{ij}^d, computes 16 measures for each link direction d, where i indexes an (x_i^R, y_i^R) component pair of the reference 4-tuple, and j indexes the 4-tuple of the neighbor in direction d. The $d^{\underline{th}}$ member of the $X_{k\ell}^R$ 4-tuple is updated as a function of the maximum valued M_{ij}^d.

$$M_{ij}^d = |G_{ij}^d| \cdot F_{ij}^d \qquad \text{where}$$

$$F_{ij}^d = 1 - \frac{|x_i^R - x_j^d| + |y_i^R - y_j^d|}{|x_i^R + x_j^d| + |y_i^R + y_j^d|} \cdot C^n \geq 0 \quad .$$

The G_{ij}^d equation differs for each link direction and will be defined below. The constant C^n varies according to the iteration number, n, and is defined to be: $C^1=1/4$, $C^2=3/8$, $C^3=3/8$, $C^4=1/2$, $C^5=1/2$, and $C^{n>5}=5/8$. The dampening factor C^n is used to reduce the penalty for large angular deviations between gradient vectors during early iterations (when angular error is greatest).

The different G_{ij}^d equations are a function of the link direction d and are as follows:

- CASE d=1, -45^o

$$G_{ij}^1 = .707\,[(x_i^R + x_j^d) + (y_i^R + y_j^d)]$$

- CASE d=2, 0^o

$$G_{ij}^2 = y_i^R + y_j^d$$

- CASE d=3, 45^o

$$G_{ij}^3 = .707\,[-(x_i^R + x_j^d) + (y_i^R + y_j^d)]$$

• CASE d=4, 90°

$$G_{ij}^{d} = -(x_i^R + x_j^d)$$

Note that in computing G_{ij}^{d}, the likelihood of an elemental edge link is optimally increased when the (x_i^R, y_i^R) pair of the reference pixel lies in the same direction as the (x_j^d, y_i^d) pair of the neighbor pixel. Also the F_{ij}^{d} acts as a dampening value that becomes smaller as the angle between the reference and link direction x,y pairs increases. When the x_i^R, x_j^d and y_i^R, y_j^d are in agreement in terms of magnitude and sign (direction), F_{ij}^{d} is nearly equal to one.

After computing the 16 values of M_{ij}^{d} for a particular direction d, the method of update of the (x_d^R, y_d^R) components of the reflectance gradient vector at the reference pixel follows. First the maximum valued M_{ij}^{d} for direction d is located. The corresponding i and j values that produced the maximum define (by indices i and j) the reference and neighbor 4-tuple pairs that are to be used to upgrade the d^{th} pair of the reference 4-tuple.

$$\text{new } x_d^R = (x_i^R + x_j^d)/2$$

$$\text{new } y_d^R = (y_i^R + y_j^d)/2$$

It may be noted that if an edge truly exists at a pixel, the iteration process causes the extent of the edge to grow as the number of iterations increases. The magnitude of the edge likelihood ratio should not change from iteration to iteration, but the variance of the ratio decreases as the number of iterations increases. As a result, because the uncertainty of the edge likelihood measure is decreased, the odds for correctly labeling a pixel with "edge" or "no edge" increase.

5. Data Display

After each iteration of the algorithm, a 4-tuple of likelihood x,y component vectors exists for each pixel of the display. Each vector corresponds to a possible edge direction. Only the largest (most likely) vector associated with each pixel is retained for display. While the decision to retain only the largest measure of likelihood is made internal to the delivered software, the pixel classification decision of "edge" or "no edge" is left to the user. Thus, the user can adjust the decision threshold as he sees fit.

The first method of data display was to return for each pixel of the image the maximum likelihood vector's magnitude, x-component, and y-component. By displaying magnitude as a color intensity on a display, the likelihood was seen as an amplitude. No edge directional data was presented. However, by displaying the x-component in one color (e.g., red) and the y-component in another color (e.g., green), edge direction and gradient up/down information could be observed concurrently with magnitude.

This color display strategy was somewhat complicated because x and y could be positive or negative; the color display understands only positive numbers. Both x and y were biased so that no negative numbers occurred. The effective zero level of a component was then assigned the midrange intensity of a given color of the display. Thus, as an example, if the x-component of a pixel's likelihood vector was assigned to the red gun of the display, x=0 resulted in a middle intensity red spot. A large positive x resulted in a high intensity red spot and a large negative x resulted in a very dim red spot. If the y-component data were treated similarly and sent to the green gun, similar results in green would be expected. When both colors were simultaneously displayed, the resultant color at a spot indicated direction. Thus, a red spot indicated a North-to-South running edge, a green spot an East-to-West running edge, and a yellow spot a N.E.-to-S.W. running edge. Note that a bright red indicated elevations rising to the East and a dim red indicated elevations falling to the East. Similar data extraction could be obtained for green and yellow.

Unfortunately, S.E.-to-N.W. edges did not show up well because of the mix of either a dim red and bright green or dim green and a bright red along the 45^{o} line.

The above display strategy had drawbacks. Aside from the difficulty of displaying direction in-appropriately when one component was negative and the other positive, the human observer found it difficult to interpret both dim and bright spots of the image as high likelihood spots. Clearly what should be done is to define the vector in terms of magnitude and angle. Then, the angle could be used to determine what mix of color guns should be used to achieve a color unique to the angle, and magnitude could establish the color intensity (i.e., the gain for each gun).

A different method for the display of edge angle is to display only the magnitudes of x and y. In this method, rather than add 128 to all x and y values (complement the sign bit) so that they are rescaled from -128:127 to 0:255, the absolute value is computed resulting in values scaled from 0 to 127. These are then multiplied by two to yield values between 0 and 255. Again, assign the resulting x and y magnitudes to the red and green guns, respectively. Color now uniquely defines the direction of an edge. Note that color used in this manner cannot indicate whether elevation rises or falls across an edge. This deficiency may be a benefit where up/down edges occur in rapid succession (e.g., along thin roads, rivers, etc.). Simply, the same color appears for the front edge and back edge pixels that lie in close proximity. (Note in contrast that the previous means of display would show a dim and bright red pixel adjacent to each other for a N-to-S edge; the average would yield the bias intensity red which is, of course, the zero-likelihood-of-an-edge red.)

Various mechanisms of data display need to be exercised to determine those useful for establishing edge threshold parameters.

6. STARAN Execution Times

The execution time is broken down into three segments: 1) computation of initial likelihood estimates, 2) an iteration upon the likelihood estimates, and 3) data display. The times given do not include the CDC 6400 execution times required to display the resulting imagery on the DIAL system.

The computation of the initial likelihood estimates is roughly equivalent to doing four 3x3 convolutions on a 512^2x8 bit pixel image. This process requires around 15 seconds of STARAN execution time. Previous performance measurements for similar processing showed that the STARAN spends over 50% of the total time waiting on I/O.

An iteration upon the likelihood estimates requires about 140 seconds of STARAN execution time. This is followed by the data display processing (a heavily I/O bound procedure) which is performed in less than 10 seconds.

7. Conclusions and Recommendations

The recommendations that follow fall into three categories, namely,

1) those aimed at upgrading the ability of the present algorithm to detect pixels that separate subregions exhibiting reflectance differences;

2) those aimed at displaying edge information in a more useful form; and, most importantly,

3) those designed to develop and implement algorithms that detect pixels that separate subregions having different subregion description vectors.

The present software loses significant capability to detect edges because measurement estimates of subregion reflectance utilize data based on very small subregions. Because the subregions are so small, the variance of the estimate of the difference between subregions tends to be high. Moreover, the estimate of the variance of the difference measure exhibits high uncertainty for the same reason - too few pixels over which to make the measure. A second factor raises the uncertainty of the variance measure. Instead of the measure being made independently for the two subregions and only thereafter combined to establish the variance of the subregion difference estimate, the estimate is made for the combined subregions. As a result,when the step between subregions is relatively large, the measure of likelihood of a pixel being an edge pixel (i.e., the ratio of the reflectance difference to the square root of the variance estimate) tends to become smaller simply because the variance measure is enlarged.

GAC suggests that two modifications to the software be made. First, software should be altered to enlarge subregions over which reflectance and variance measures are made. Secondly, the software should be changed so that the estimate of the variance of the sub-

region step difference more nearly represents the true variance in regions of edges. This change should incorporate a procedure that independently estimates the variance of each of the subregions before the data is combined to develop the estimate of the variance of the subregion step difference.

Prior to instituting the changes above, GAC suggests performing particular experiments aimed at indirectly verifying that the variance estimate of the difference measure is inadequate. In particular, GAC suggests pre-processing imagery (prior to processing with the edge detection software) so as to create a scene that has the equivalent of the same variance estimate everywhere. With such a scene, a "normalized" scene, the impact of the denominator (the variance of the subregion difference step estimate) of the pixel edge likelihood ratio will be diminished.

The initial processing within the edge detection software would utilize step derivative reference patterns rather than step reference patterns when treating a normalized scene.

GAC finds no significant fundamental problems with the iterative subregion growing processes of the present algorithm. Yet, they should be made more efficient and an attempt should be made to allow region growing (or edge element extension) to proceed in larger steps in later iterations. With the present procedure, each successive iteration links and treats a smaller amount of new data. After N iterations, only about 1 part of data of N+1 parts will be new (independent) during the processing associated with the succeeding iteration.

To be able to observe the results of edge detection processing quickly, it is recommended that new display capabilities be implemented. At least the capability to display elemental edge directions as colors and the likelihood ratio magnitude as a color intensity should be implemented.

A severe limitation of the present algorithm is that it requires that neighborhood subregions exhibit different reflectance values in order for an "edge"/"no-edge" decision to be made. But different subregions with the same average reflectance value can and do exist; demarcations between such regions can't be detected. GAC suggests that the algorithm be re-structured so as to detect demarcation lines of subregions that have different description vectors rather than just scalars. The difference between subregions would be described as a vector operation, potentially, the likelihood ratio would be developed directly as in a "Maximum Likelihood" procedure. Not only could such an algorithm be used to detect demarcations between subregions with no reflectance difference, it could be used to detect edges between regions with specific characteristics. Because vector data would be used, uncertainty would be reduced.

Development of the algorithm would address the issue of the spatial relationships between a pixel that is an edge pixel contender, the locations of subregions, and the shapes of the sub-regions.

The results of work using the present algorithm show that it is not the "edge" detection answer. This was largely recognized at the start of the program. Results, however, have suggested the direction of future study.

B. CDC6400:STARAN I/O Buffer Formats

The order or sequence of events across the interface is as follows:

1. A buffer of 120 CDC words: 225 STARAN words containing the convolution filter data and parameters is sent to the STARAN.

2. The image data (512^2x8-bit pixels) is sent to the STARAN in buffers containing 30 scan lines formatted as follows: 72 CDC words: 135 STARAN words per scan line, or 2160 CDC words: 4050 STARAN words per buffer. Eighteen buffers are sent to the STARAN: the first 17 contain 30 scan lines each, and the last buffer contains 2 scan lines (all buffers are the same size).

   After the first buffer of 30 scan lines is sent to the STARAN, the following sequence is repeated 17 times:

   - send buffer of 30 scan lines to the STARAN
   - receive 10 buffers (each contains 3 "scan lines") from STARAN

   followed by one last input of a buffer containing 2 "scan lines" from STARAN. The buffers containing 3 "scan lines" received from the STARAN contain the probability estimates. Each buffer contains 1640 CDC words: 3075 STARAN words. The STARAN data is packed solidly into the first 3072 words: the last three words are spare in order to satisfy the multiple of 15 requirement for all buffer sizes. The probability data for a given image pixel looks like (1024 STARAN words for a 512 8-bit pixel scan line)

	0 7	8 15	16 23	24 31
STARAN Word 1	X1	Y1	X2	Y2
STARAN Word 2	X3	Y3	X4	Y4

   where each item is a signed 8-bit "probability" value with six fractional bits.

The edge directions represented by the X_i, Y_i pairs are

X1,Y1	-45^o
X2,Y2	0^o
X3,Y3	45^o
X4,Y4	90^o

3. At this stage, the I/O that takes place is dependent upon the value of ITBDSP, the number of iterations between displays. If ITBDSP is equal to one, then the initial probability estimates just computed are converted to the pixels of a displayable image. Otherwise one or more iterations take place, followed by the displayable image process. Once the "first time" exception to I/O is taken care of, the normal procedure is to iterate ITBDSP times, followed by the displayable image process. The normal I/O procedure is described.

   The CDC transmits 171 buffers of probability data to the STARAN, and receives 171 buffers of updated probability data back. Each buffer contains 1640 CDC words: 3075 STARAN words as described above in step #2. The I/O procedure first transmits one buffer of data to the STARAN followed by repeating the following two steps 170 times:

   - send buffer of probability data (3 "scan lines") to STARAN
   - receive buffer of updated probability data (3 "scan lines") from STARAN

   Finally, the last buffer of updated probability data (2 "scan lines") is received from the STARAN. Note that the last buffer sent to the STARAN also contained only 2 "scan lines" worth of data. Also note that if the last word of the first buffer is non-zero, the STARAN interprets this as the termination signal and quits. This entire step #3 is repeated ITBDSP times.

4. The I/O procedure to obtain a displayable image from the STARAN is essentially the reverse of step #1. An exception to this is the fact that STARAN sends back three displayable images: the R image, X image, and Y image where the pixels of each are related by the equation

$$R = \sqrt{X^2 + Y^2} \quad .$$

   In other words, the X and Y image pixels are simply the components of the gradient vector with the largest R value at a given pixel position.

First a buffer of three "scan lines" of probability data is sent to the STARAN (1640 CDC words: 3075 STARAN words). Then the following steps are repeated 17 times:

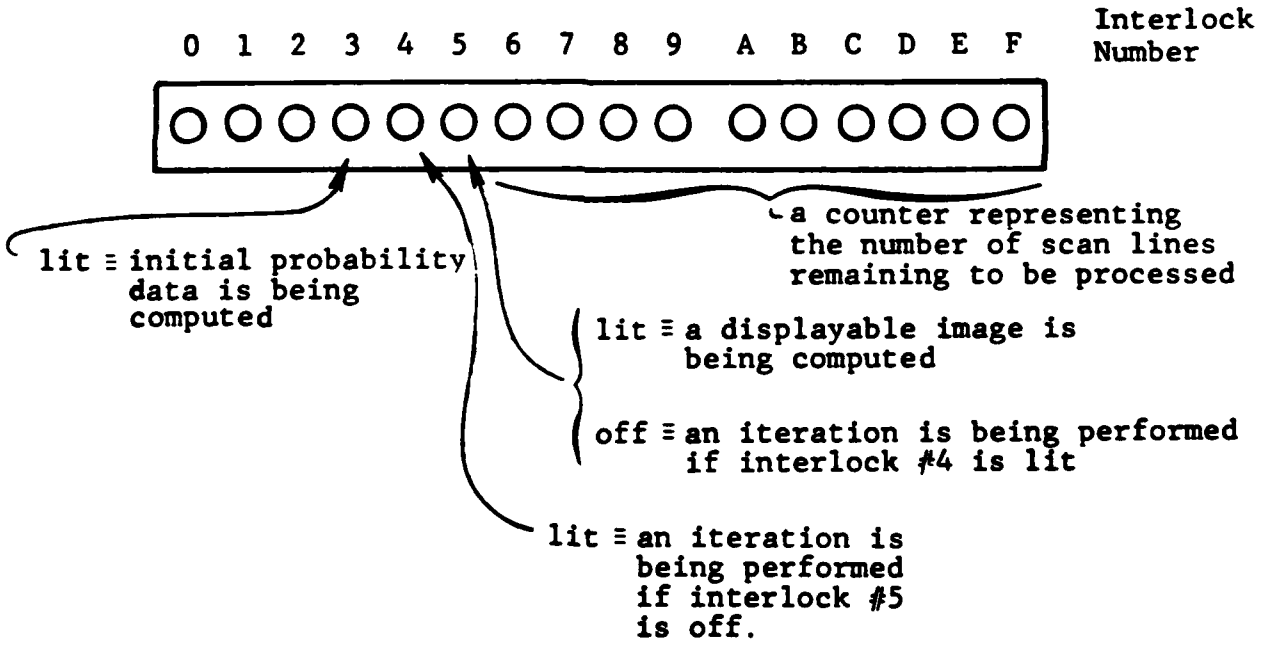
- send 10 buffers of probability data (3 "scan lines" each) to the STARAN.
- receive a buffer of 30 scan lines of pixel data for the R image
- receive a buffer of 30 scan lines of pixel data for the X image
- receive a buffer of 30 scan lines of pixel data for the Y image

Finally, three more buffers of pixel data (2 scan lines each) are received from the STARAN to complete the I/O process.

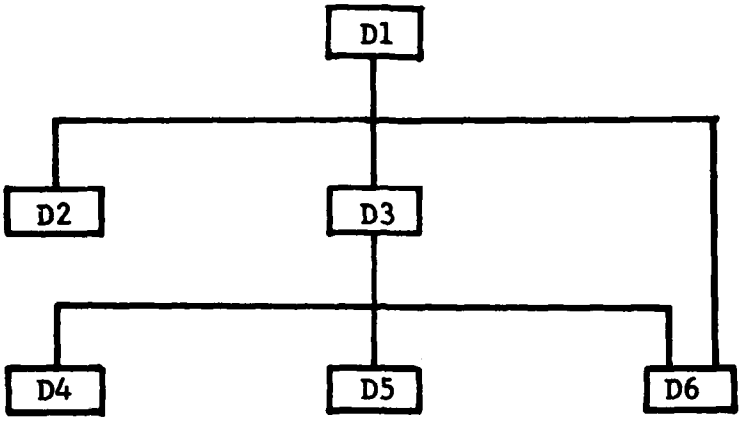
Steps 3 and 4 are repeated for all subsequent iterations until the DIAL user signals a terminate command.

C. STARAN Interlock Usage

The 16 interlock lights are used as a display for the relaxation labelling software. Interlocks 0 and 1 are used internally by the I/O subroutines to signal full and empty I/O buffers. Interlock #3 is set whenever the initial probability computation software is executing. Interlock #4 remains set while the iteration: displayable image software is resident and executing. Interlock #5 is set during the displayable image computation process. Interlock lights 6 thru F (lit ≡ a binary 1, off ≡ a binary 0) contain the number of scan lines remaining to be processed. This value is decremented from 511 down to 0 during a process.



D. CDC6400 FORTRAN SUBROUTINES



Page	Program Module
D1	EDGERLX - Main Controlling Executive Module
D2	LDSTAR - Load and Execute STARAN Program
D3	RELAX - Compute Initial Probability Estimates
D4	SENDIM - Send Image to STARAN
D5	GTPROB - Get Probability Estimates from STARAN
D6	ITRATE - Perform an Iteration

EDGERLX - Main Controlling Executive Module

This program was written by ETL to utilize the FORTRAN callable relaxation labelling edge detection subroutines written by GAC. It calls the RELAX subroutine to generate the initial set of probability estimates. The ITRATE subroutine is then repeatedly called for all subsequent iterations.

The interface between this routine and the RELAX and ITRATE subroutines is described in section H, Program Usage.

LDSTAR - Load and Execute STARAN Program

This subroutine is called to 1) attach the current job to the STARAN, and 2) load a specified program in the STARAN. The STARAN program is one that automatically begins executing when loaded. e.g.,

```
CALL  LDSTAR(NAME,IUIC,IACTION,ISTAT)
```

INPUTS:

1. NAME is the name of the STARAN relaxation routine. It is a character constant and should be

   10HRELAX␣ALD␣

2. IUIC is the "User Identification Code" specifying where on the STARAN's DK1 disc drive the file NAME is located. For relaxation routines, IUIC should equal 10001B.

3. IACTION is an integer value set equal to 1 or 0: If 0, then wait until the STARAN is finished if the status indicates it is attached to another job. If 1, then exit if the STARAN is attached to another job.

PROCESS:

1. attach the STARAN if on-line and available
2. load the specified STARAN program

OUTPUTS:

1. ISTAT is an integer status value indicating:

   0 = the STARAN is ready and OK.
   1 = the STARAN is not on-line.
   2 = the STARAN is attached to another job.
   3 = unable to load the STARAN program specified by NAME and IUIC.

CALLED BY: a user main program

SUBROUTINES CALLED: none

RELAX - Compute Initial Probability Estimates

This routine begins the relaxation process by transferring a buffer of control parameters and filter values to the STARAN. The source image is transferred to the STARAN and the initial probability estimates received. The ITRATE subroutine is then called to perform the requested number of iterations followed by computation of displayable images.

INPUTS:

1. A real array of filter values preceded by their dimension (currently must be 3x3 filters).
2. A normalization value which can be used to provide a common normalization value for the entire image while computing the initial probability estimates, NORMLZ.
3. The number of iterations between displays, ITBDSP.
4. A threshold value used to zero pixels of the displayable images, ITHRSH.
5. A named COMMON/INFO/ containing the following image descriptors:

   a) IMAGNM(4) - the name of the original image.
   b) IPROBS(4) - the name of the probability data image.
   c) NAME(4) - the name of the updated probability data image.
   d) NAMR(4) - the name of the resultant image.
   e) NAMX(4) - the name of the X-component image.
   f) NAMY(4) - the name of the Y-component image.
   g) LABEL(2602) - the label header descriptor for a 512^2 x 8 bit pixel image.
   h) LABELP(2602) - the label header descriptor for a 171 x 1640 x 60 bit pixel probability data image.

PROCESS:

1. ITBDSP is set to one if less than or equal to zero.
2. NORMLZ must be >0 and <256.
3. ITHRSH must be >0 and < 256.
4. The first value, N, of the filter array must be equal to 3.

5. The average value of each filter is computed.

6. The standard deviation of each filter is computed. It must not be equal to zero.

7. Each filter element is modified by subtracting the average.

8. The maximum number of bits of precision of the largest filter element of each filter is computed.

9. The parameters and filter data are output to STARAN.

10. A "CALL FIND" on IMAGNM is executed so that it can be DREAD after it has been displayed by the main routine.

11. A loop is entered to repeatedly send image data to the STARAN and receive probability data in return.

12. The IPROBS file of probability data is DCLOSE'd, LOCATE'd, and LBLRD so that it can be DREAD by the ITRATE subroutine.

13. ITBDSP is decremented since the initial probability data is considered to be the result of a "first iteration."

14. ITRATE is called to perform ITBDSP iterations followed by developing displayable images.

OUTPUTS:

1. A DIAL "image" of probability estimates called IPROBS.

2. Three displayable images called out in the named COMMON as NAMR(4),NAMX(4), and NAMY(4).

SUBROUTINES CALLED: SENDIM,GTPROB,ITRATE

CALLED BY: EDGERLX main program

SENDIM - Send Image to STARAN

This subroutine assembles 30 scan lines of a 512^2 x 8 bit pixel image into an array using 72 CDC words per scan line.

INPUTS:

1. IMAGNM(4) contains the name of a DIAL image that can be DREAD.
2. J is a number representing the J^{th} time this subroutine has been called.
3. INRECS is in a COMMON and should have been initialized to 512 for the total number of scan lines this subroutine will attempt to DREAD.

PROCESS:

1. a loop is repeated 30 times (or until INRECS goes to zero) while 30 scan lines of imagery are DREAD into a buffer.
2. the buffer of 30 scan lines is output to the STARAN.

OUTPUTS:

1. INRECS is decremented by 30.
2. a buffer of 30 scan lines is output to the STARAN.

CALLED BY: RELAX

GTPROB - Get Probability Estimates from STARAN

This subroutine reads 10 buffers of probability data from the STARAN.

INPUTS:

1. IPROBS(4) contains the name of a DIAL image that can be DWRITE'n.
2. K is a value representing the K^{th} time this routine has been called.

PROCESS:

1. a loop is entered wherein 10 buffers (each contains 3 scan lines) of probability data are read from the STARAN and written onto a DIAL image file.

OUTPUTS:

1. 10 more "scan lines" of probability data from the STARAN are added to the IPROBS DIAL image.

CALLED BY: RELAX

ITRATE - Perform An Iteration

This subroutine performs ITBDSP iterations followed by obtaining a displayable result image.

INPUTS:

1. ITBDSP represents the number of iterations between displays. If equal to -1, the STARAN is commanded to terminate processing. If equal to zero, no iterations are performed and displayable result images are obtained.

2. The named COMMON /INFO/ area contains:

   a) IMAGNM(4) - the name of the original image.
   b) IPROBS(4) - the name of the probability data image.
   c) NAME(4) - the name of the updated probability data image.
   d) NAMR(4) - the name of the resultant image.
   e) NAMX(4) - the name of the X-component image.
   f) NAMY(4) - the name of the Y-component image.
   g) LABEL(2602) - the label header descriptor for a 512^2 x 8 bit pixel image.
   h) LABELP(2602) - the label header descriptor for a 171 x 1640 x 60 bit pixel probability data image.

3. IPROBS must be in a state wherein it can be DREAD.

4. NAMR, NAMX, and NAMY must be in a state wherein they can be DWRITE'n.

PROCESS:

1. If ITBDSP is zero, then displayable images are obtained. Go to step 4.

2. If ITBDSP is negative, a minus one is stored into the last word of a buffer sent to the STARAN to signal the STARAN to terminate.

3. If ITBDSP is greater than zero, the following loop is entered and repeated ITBDSP times.

a) a DIAL image called NAME is KREATE'd and LBLWRT'en in order to hold the updated probability data from the STARAN.

b) the following steps are repeated 171 times:

- DREAD IPROBS data (3 scan lines)
- SWRITE IPROBS data to STARAN
- SREAD NAME data from STARAN (3 scan lines)
- DWRITE NAME data

c) the updated probability file NAME is DCLOSE'd.

d) the old probability file,IPROBS, is DPURGE'd and RETURN'd to the system.

e) the updated probability file NAME is LOCATE'd, LBLRD, DRENAME'd, and called IPROBS.

f) the updated probability file IPROBS is then LOCATE'd and LBLRD so that it can be subsequently accessed by DREAD commands.

4. The following steps are repeated 18 times:

a) the following steps are repeated 10 times:

- DREAD IPROBS data (3 scan lines)
- SWRITE IPROBS data (3 scan lines)

b) SREAD 30 scan lines of NAMR data and DWRITE them.

c) SREAD 30 scan lines of NAMX data and DWRITE them.

d) SREAD 30 scan lines of NAMY data and DWRITE them.

5. The NAMR, NAMX, and NAMY images are DCLOSE'd.

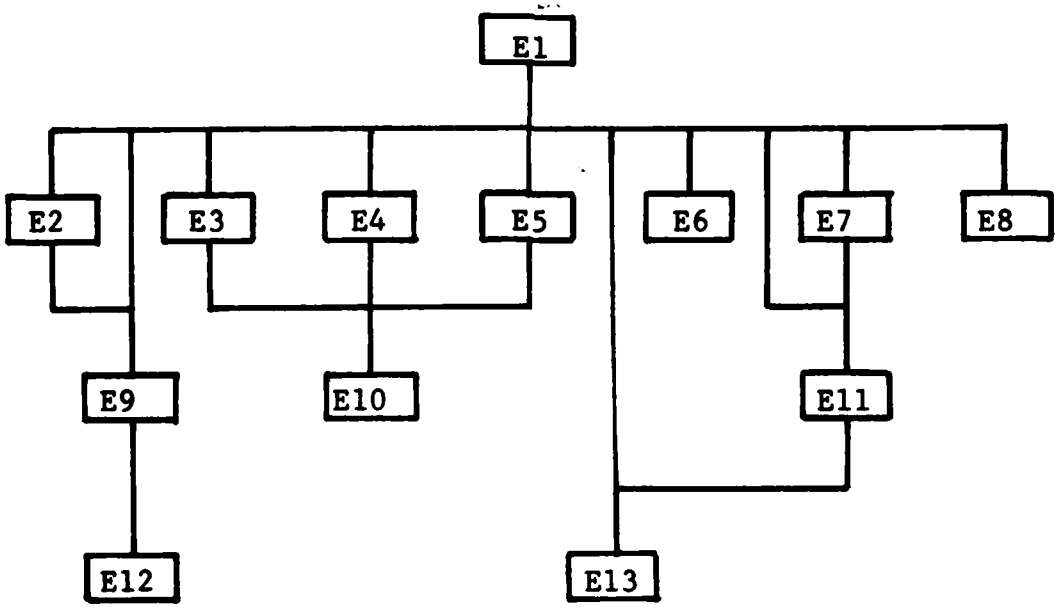
OUTPUTS:

1. An IPROBS file of probability data that can be DREAD.

2. NAMR, NAMX, and NAMY files of image data that must first be LOCATE'd before calling IMGDSK to display them.

CALLED BY: RELAX,EDGERLX

E. STARAN PROBABILITY ROUTINES



Page	Program Module
E1	EXEC - Main Routine For Probability Computations
E2	INITARRAY - Initialize Array Memory
E3	AVERAGEX - Compute NxN Averages For Image
E4	STANDEVX - Compute NxN Standard Deviations For Image
E5	CONVOLUTN - Compute NxN Convolutions For Image
E6	TOPBOTTOM - Handle Top and Bottom Border Values
E7	RIGHTLEFT - Handle Right or Left Border Values
E8	SETILOCKS - Display Value in Interlock Lights
E9	BULK2ARAY - Load Array From Bulk Memory
E10	SHIFTDOWN - Shift Down Between Array Boundaries
E11	ARAY2BULK - Load Bulk Memory From Array Memory
E12	MOVELEFT - Position Scan Lines In Array Memory
E13	CDCIO - Keep I/O Busy In Background

EXEC - Main Routine for Probability Computations

The EXEC is the controlling routine in the STARAN during the initial probability computation that edges lie in particular directions. Once completed, the iteration software is loaded as an overlay and remains in control until a termination signal is received from the DIAL user via the FORTRAN subroutine ITRATE.

INPUTS:

1. A 225 word buffer containing parameters and filter data. The probability executive and support routines have been programmed to compute edge probability values for four 3x3, six 4x4, or eight 5x5 convolution filters. However none of the other software modules will support anything but the four 3x3 filters. The fixed size of 225 words was selected to accommodate the largest case, eight 5x5 filters and parameters. The format is as follows:

	0 7	8 15	16 23	24 31
WORD 1:	NORMLZ	N=3	ITBDSP	ITHRSH
2	filter #1 precision		0	
3	F1(1,1)		0	
4	F1(1,2)		0	
⋮	⋮		⋮	
10	F1(3,2)		0	
11	F1(3,3)		0	
12	σ_{F1}		0	
13	filter #2 precision		0	
14	F2(1,1)		0	
15	F2(1,2)		0	
⋮	⋮		⋮	
21	F2(3,2)		0	
22	F2(3,3)		0	
23	σ_{F2}		0	
24	filter #3 precision		0	
⋮	⋮		⋮	

Words 2, 13, 24, and 35 contain the number of bits of precision required to represent the largest filter element for filter #1, 2, 3, and 4 respectively. Each value of a filter contains 8 fractional bits. Filter values must be in the range

$$|Fn(i,j)| \leq 127 \frac{255}{256}$$

Words 3-11, 14-22, 25-33, and 36-44 contain the filter elements of the four 3x3 filters respectively. Words 12, 23, 34, and 45 contain the value of the standard deviation, σ_n, of each of the four filters. σ_n is a 16 bit value with eight fractional bits.

PROCESS:

1. Based on the size of the filter (N should equal 3), all initializations are performed by using the value of N as an index into various tables.

2. The first buffer of 30 scan lines is read and the second buffer requested.

3. The array is initialized and the values of the left border generated and output.

   Next the major execution loop is entered.

4. BULK2ARAY is executed to move a scan line from an input buffer to array memory.

5. CDCIO is called between all calls to major subroutines to keep I/O busy in a background mode.

6. AVERAGEX is called to compute the average pixel value of all NxN image patches (512 - (N-1)/2 = 510 averages simultaneously).

7. STANDEVX is called to compute the standard deviation value of all NxN image patches - 510 simultaneously.

8. CONVOLUTN is called to compute the convolution value of all NxN image patches (510 simultaneously) for all 2(N-1) filters.

9. TOPBOTTOM is called to generate values for the top and bottom borders.

10. ARAY2BULK is called to transfer a "scan line" of probability values from array memory to an output buffer.

11. Loop counts are checked and a branch back to step #4 is made if there are more scan lines to process.

12. RIGHTLEFT is called to generate values for the right border.

13. The last buffer of data is output (partially filled), and the iteration software is loaded and executed.

OUTPUTS:

1. 512 scan lines of probability data (the equivalent of 8 images worth of data) are generated and sent to the CDC for interim storage.

INITARRAY - Initialize Array Memory

This subroutine is called to initialize array memory prior to the initial probability calculations.

INPUT:

N, the dimension of the filters.

PROCESS:

This routine loads the first N-1 scan lines into array memory. The next time a scan line is loaded during the major execution loop of the EXEC, N scan lines will be present - enough to perform the associated convolution calculations.

OUTPUT:

An array that is ready for the major execution loop of the EXEC.

CALLED BY: EXEC

SUBROUTINES CALLED: BULK2ARAY

AVERAGEX - Compute NxN Averages For Image

This subroutine computes the average pixel value for all NxN windows of the current center scan line.

INPUTS:

1. ADDXXXES is a table of add field parameters indexed by N.
2. NSQINVRSE is the inverse of the number of elements in an NxN window (scaled 16).
3. NUMROWADD is a count of the number of rows to add together, equal to N-1.

PROCESS:

1. The first two scan lines (leftmost) are added together.
2. A loop is executed to add additional scan lines to the accumulating sum of the first two.
3. The row sums are added together.
4. The sum of all pixels in an NxN window is multiplied by $1/N^2$ to form the average value.

OUTPUTS:

1. The average value for all NxN windows of the center scan line, 16 bits scaled 8.

CALLED BY: EXEC

SUBROUTINES CALLED: SHIFTDOWN

STANDEVX - Compute NxN Standard Deviations For Image

This subroutine computes the standard deviation value for all NxN windows of the current center scan line.

INPUTS:

1. N, the dimension of the window.
2. NSQUARED, the value of N^2.
3. MEWSUBX, the average pixel value for all NxN windows.

PROCESS:

The standard deviation equation

$$\sigma_x = \sqrt{\sum_i^N \sum_j^N X_{ij}^2 - N^2 \bar{\mu}_x^2}$$

is computed, where $\bar{\mu}_x$ is the average value of the pixels in the window (computed in AVERAGEX subroutine). The array field, SIGMASUBX, contains the standard deviation values which are 16 bits, scaled 4.

OUTPUTS:

1. A field of standard deviation values, SIGMASUBX, for all NxN windows of the current center scan line. The values are 16 bits, scaled 4.

CALLED BY: EXEC

SUBROUTINES CALLED: SHIFTDOWN

CONVOLUTN - Compute NxN Convolutions For Image

This subroutine computes a convolution value (510 simultaneously) for each pixel in a scan line. The process is repeated 2(N-1) times for each of the 2(N-1) filters.

INPUTS:

1. buffer of filter values formatted as shown in the description of the probability EXEC.
2. a mask, SCONVMASK, to exclude the (N-1)/2 border pixels from generating convolution values.
3. the number of bits (precision) required to hold the largest filter element.
4. the standard deviation value of each filter.

PROCESS:

1. initialize STOREPROB, a table index of where to store the 2(N-1) convolution values; FILTERPTR, a table index of the buffer of filter values; and CFILTERS, a count of the number of filters to process.

   Major loop for each filter to process.

2. compute the sum of the products of each filter element with each corresponding image pixel, 510 are computed simultaneously.
3. normalization of the result is performed by:
   a) ruling out divide by zero of σ_X, the standard deviation of an NxN portion of the image, if zero.
   b) form the product of the filter and image standard deviations, the normalization divisor.
   c) if the optional NORMLZ value has been selected by the DIAL user, divide by it rather than the product of the standard deviations.

d) divide and clear to zero all quotients that overflowed.

4. Make sure the values produced are $-2 \leq p \leq 1\frac{127}{128}$.

5. Store the values into a holding area.

6. Loop back to step #2 if there are more filters.

OUTPUTS:

1. 2(N-1) probability values for 512-(N-1) pixels of a scan line.

CALLED BY: EXEC

TOPBOTTOM - Handle Top and Bottom Border Values

This subroutine stores a probability value (arbitrarily selected to be 1/4) into the top and bottom (N-1)/2 pixel positions.

INPUTS:

1. N, the size of the convolution filters.
2. NFILTS, the number of filters.
3. MEDIOCRE, a constant containing four probability values equal to 1/4.
4. NUMARRAYS, the number of associative array memory modules being used.

PROCESS:

1. the number of filters (is the same as the number of probability values generated) is multiplied by 8 to obtain the number of bits required to hold the probability values generated - each value is 8 bits with 6 fractional bits.
2. the MEDIOCRE probability values are stored into array memory into the first N-1 words.
3. all probability values are shifted up (N-1)/2 positions to center the values.

OUTPUTS:

1. centered probability values for a scan line ready for output.

CALLED BY: EXEC

RIGHTLEFT - Handle Right or Left Border Values

This routine is called to output (N-1)/2 scan lines of probability values for the right and left borders of the image. The probability value of 1/4 was arbitrarily selected as the value for all edge directions.

INPUTS:

1. N, the size of the convolution filters.
2. NFILTS, the number of filters.
3. MEDIOCRE, a constant containing four probability values equal to 1/4.
4. FIRSTIME, a flag used when N=4 in order to output 2 "scan lines" worth of probability values for the right border (only one line is output for the left).

PROCESS:

1. the number of filters (is the same as the number of probability values generated) is multiplied by 8 to obtain the number of bits required to hold the probability values generated - each value is 8 bits with 6 fractional bits.
2. the MEDIOCRE probability values are stored into array memory, and the ARAY2BULK routine is called (N-1)/2 times.

OUTPUTS:

1. (N-1)/2 "scan lines" of MEDIOCRE valued probability data in an output buffer.

CALLED BY: EXEC

SUBROUTINES CALLED: ARAY2BULK

SETILOCKS - Set Interlock Lights

This subroutine is called by the EXEC to display a binary value in the common register in the interlock lights. This is used to display the number of scan lines remaining to be processed.

INPUTS:

1. the lower half of the common register contains the value to display.
2. the BL register contains the number of interlock lights required to display the value.

BULK2ARAY - Load Array From Bulk Memory

This subroutine is called to transfer a scan line of pixels from an input buffer to array memory.

INPUTS:

1. SCANPOINT references the next scan line in the input buffer.
2. SCANSLEFT, a double counter, contains both the total number of scans left to load, and the number of scan lines left in the current input buffer.

PROCESS:

1. Both counters in SCANSLEFT are decremented.
2. The scan line is loaded into the input area of array memory.
3. SCANPOINT is updated by adding 135 (each scan line requires 128 words plus 7 words of pad in order to fulfill the multiple of 15 requirement).
4. The mixed mode input data is unscrambled and moved into place.

OUTPUTS:

1. The leftmost (oldest) scan line is overlayed by moving the other scan lines (including the new one) left one scan line position.
2. SCANPOINT and SCANSLEFT are updated.

CALLED BY: EXEC.INITARRAY

SUBROUTINES CALLED: ORDER$,MOVELEFT

SHIFTDOWN - Shift Down Between Array Boundaries

This module contains two subroutines for between array boundary shifting: SHIFTDOWN and SHIFTUP. The SHIFTUP subroutine was later dropped from being required. Only the SHIFTDOWN subroutine is described.

INPUTS:

1. The data to be shifted must be in the last 32-bit field of array memory: bits 224-255.

2. NUMARRAYS, the number of array memory modules being used - the number of array boundaries to shift across is determined from this value.

PROCESS:

The data in the last 32-bits of array memory is downshifted one word position, a 32-bit value of zero is stored into the top.

OUTPUT:

The last 32-bit field of data in NUMARRAYS array memory modules is shifted down one word position.

CALLED BY: AVERAGEX,STANDEVX,CONVOLUTN

ARAY2BULK - Load Bulk Memory From Array Memory

This subroutine is called to transmit a "scan line" of probability values from array memory to an output buffer.

INPUTS:

1. XYFACTORS are used to generate the X and Y components of the gradient or probability values.
2. ENDARRAY is used to flag access beyond the first two associative array memory modules.
3. NSETSDIRS is the number of sets (a set contains 2(N-1) directional probability values for each pixel) of probability values that will fit in an output buffer.
4. OUTBUFPTR is the address of the next location in an output buffer.
5. WHICHOUTB is a data structure indicating both the output buffer currently being output, and the next available empty output buffer. See CDCIO description for more detail.

PROCESS:

1. The individual, 2(N-1), probability measures for each pixel are converted to X,Y components.
2. The current output buffer status is checked: if not available, CDCIO is called until the next one has been output and is therefore empty and available.
3. The number of 32 bit words per set of directions is determined.
4. A loop is executed until either 512 sets of directions of a scan line have been output, or the number of sets of directions in an output buffer has been reached, i.e., the output buffer is full.

Case 1 - all 512 sets of directions of a scan line have been output and the buffer is not full.

5. The current output buffer pointer, OUTBUFPTR, is saved; and the remaining number of sets of directions that will fit in the current output buffer, NSETSDIRS, is saved.

Case 2 - an output buffer is full.

6. The current output buffer status is set to indicate full.

7. The next output buffer is selected to be the current one.

8. CDCIO is called to initiate an output in case thats what it is waiting to do.

9. The NSETDIRS count is re-initialized and OUTBUFPTR is initialized for the new output buffer.

OUTPUTS:

1. a scan line (512 sets of probability X,Y components) is transmitted from array memory to an output buffer.

2. OUTBUFPTR and NSETSDIRS are updated.

CALLED BY: EXEC,RIGHTLEFT

SUBROUTINES CALLED: CDCIO

MOVELEFT - Position Scan Lines In Array Memory

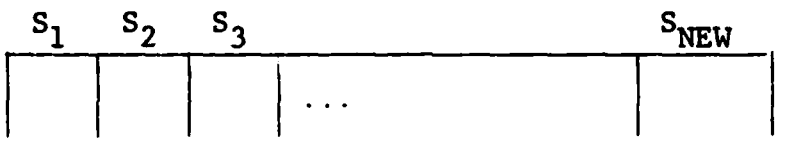
This subroutine is called to move all scan lines left one scan line position, overlaying the leftmost (oldest) scan line.

INPUTS:

1. NUMTOMOVE, the number of bits occupied by N-1 contiguous scan lines.

PROCESS:

The format of array memory looks like



where S_1 is scan line #1, S_2 is scan line #2, etc. S_{NEW} represents the newly loaded input scan line (BULK2ARAY). This routine performs the following:

$$S_i \leftarrow S_{i+1}$$

$$S_N \leftarrow S_{NEW}$$

OUTPUT:

An array memory with all scan lines moved left one scan line position.

CALLED BY: BULK2ARAY

CDCIO - Keep I/O Busy in Background

This subroutine was written to support the overlap of I/O in a background mode with execution. A table of I/O functions is cycled to direct what is done next. There are three return points to indicate various status conditions pertinent to the routines that call CDCIO.

e.g., Calling sequence:

```
BAL,R5    CDCIO
```

- busy return
- an empty output buffer is available
- waiting on an output buffer to be filled

The first return is used whenever the PDP11 is busy transferring I/O data between STARAN and the CDC. The second return is used to signal the ARAY2BULK routine that an empty output buffer is now available so that it can proceed by transferring results to the empty output buffer. The third return is used by CDCIO to notify the EXEC on the last transfer that it cannot proceed with the output until the buffer is filled.

INPUTS:

1. WHATNEXT is the I/O function jump table index.

2. WHICHOUTB is a data structure that indicates the status of the 4 output buffers and is formatted as follows:

WHICHOUTB:	0 7 BUFFER TO OUTPUT INDEX	8 15	16 31 BUFFER TO FILL INDEX
+1	X	Y	F000
+2	X	Y	E000
+3	X	Y	D000
+4	X	Y	C000

{output buffer starting addresses

$X \neq 0$ the output buffer is full and ready to be output.

$X = 0$ the output buffer is not full.

$Y \neq 0$ the output buffer is currently being output.

$Y = 0$ there is no I/O activity in the output buffer.

The first word of WHICHOUTB contains table index values into the WHICHOUTB table. The index values cycle between one and four. The upper index is used by the output portion of CDCIO. The lower index is used by the ARAY2BULK subroutine to obtain the address of the next available output buffer.

3. NUMINBUFS is a count of the total number of input buffers to read from the CDC6400.

4. STARTOVER is the restart jump table value used to recycle WHATNEXT. Currently for the 3x3 filter case

STARTOVER is equal to 22 so that there are 10 output buffers for every input.

PROCESS:

The routine repeatedly cycles through 22 I/O steps until the value of NUMINBUFS has been decremented to zero and it is waiting on the last output buffer to be filled so that it can issue the output command. The code for the four I/O functions (OUTPUT, WAITOUTPT, INPUT, WAITINPUT) is fairly simple and straightforward.

OUTPUTS:

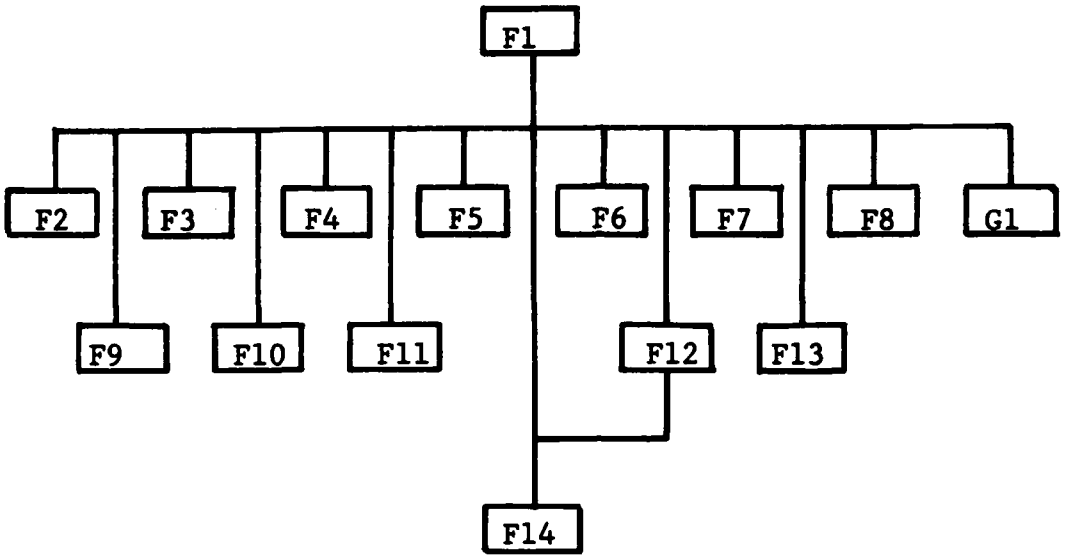
1. WHATNEXT is bumped whenever an I/O function has completed.

2. NUMINBUFS is decremented whenever an input command is issued.

3. Output buffer status in WHICHOUTB is set to busy when I/O is taking place, and set to empty after output has occurred. Also the upper index in WHICHOUTB is cycled after an output has completed.

CALLED BY: EXEC,ARAY2BULK

F. STARAN ITERATION ROUTINES



Page	Program Module
F1	EXEC - Main Routine For Iteration Computations
F2	NEXTCOL - Load Array Memory From Bulk Memory (BULK2ARAY)
F3	GFUNCTION - Compute G Component of Probability Update Equation
F4	FFUNCTION - Compute F Component of Probability Update Equation
F5	FTNGXFTNF - Compute Product of F and G Components
F6	B2ABEST - Bulk to Array Best Correlation Values
F7	UPDATE - Update Probability Values
F8	A2BBEST - Array to Buffer Best Correlation Values
F9	MOVE - Move Reference Data For Next Correlation
F10	BEST - Select Best Correlation Values
F11	MOVENEWXY - Move Updated Values From Bulk to Array
F12	A2BUPDATE - Array to Buffer Update Values
F13	ALIGN - Align Link With Reference Data
F14	CDCIO - Keep CDC I/O Busy In Background

EXEC - Main Routine for Iteration Computations

The EXEC is the controlling module for the relaxation calculations.

INPUTS:

1. X:Y component probability estimate values
   There are 4 pairs of X:Y values. Each pair of values corresponds to a link direction.
2. THRESHOLD value between Ø and 255.
3. ITBDSP - the # of iterations between displays
4. SCANIN - contains the # of total scanlines, and the number of scanlines per buffer.

PROCESS

1. many initialization functions are performed
   a. the # of words to read (3 scanlines of data) from the CDC.
   b. the input and output TRANBLOCK'S are initialized to the I/O buffer addresses
   c. the first buffer of data is read and waited for.
   d. INBUFPTR and OUTBUFPTR (contains the address of input buffer and output buffer) are defined for the next call and update routines.
   e. last word of input buffer is checked, if not zero, then exit.
   f. next buffer of data is requested from CDC
   g. interlock #Ø and #1 are reset (cleared)
   h. further initializations are performed that establish link, group and shift controls that are used by many different modules during the computations of the components of the correlation values.
2. a) to load the arrays with 1 scanline of data, NEXTCOL is executed.

   b) to load the arrays with the next scanline of data, NEXTCOL is executed. It is necessary to have 2 scanlines of data in the arrays to do any calculations.

3. to keep I/O busy in the background subroutine, CDCIO is executed between all calls to major subroutines.

4. GFUNCTION is executed, and calculates one half of the correlation equation.

5. FFUNCTION is executed, and calculates the second half of the correlation equation.

6. correlation value is computed by executing FTNGXFTNF.

7. the previous best correlation values are loaded into the array from temporary storage by executing B2ABEST.

8. the current best correlation values are compared to the previous best correlation values and the larger of the two is saved when BEST is executed.

NOTE: If this is the first of M iterations in the particular link direction, steps 7 and 8 are not executed, and execution continues with step 9.

9. A2BBEST is executed to temporarily save the new current best correlation values.

10. reference data is shifted left-end around by executing MOVE. This allows getting all 16 possible group combinations of reference for a particular link. If all possible combinations have not yet been computed steps 2b - 10 are repeated.

11. the final best correlation values are loaded into the arrays from the temporary storage by executing B2ABEST again.

12. reference data X:Y values for one link direction are updated and moved to a temporary buffer when UPDATE is executed. If all 4 link directions have not been updated, steps 2b -12 are repeated.

13. All 4 link directions have been updated. All the updated X:Y values are loaded into the arrays to correct the ordering of the pairs by executing MOVENEWXY.

14. A2BUPDATE is executed to send all updated X:Y pixel values for reference scanline to output buffer. A2BUPDATE sets Interlock #Ø to signify that an output buffer is full and ready to go. The CDCIO routine is executed to issue the output command, and a loop back to step # 2b is executed.

15. Interlock #1 is used to signal that another input buffer is available.

16. When the last buffer of data is being output a wait until output is completed is executed.

When ready, the DSPLAYEXC routine is called and all

routines to prepare a displayable image are executed. The image is output to the CDC and control is returned to EXEC and it is ready to begin another iteration if commanded.

OUTPUTS:

1. updated X:Y component probability values(4 pairs of X:Y values, one pair for each link direction).

   these updated X:Y values are used for succeeding iterations.

   DSPLAYEXC uses the values, (does not destroy them) to determine corresponding pixel values for a displayable image.

NEXTCOL - Load Array Memory From Bulk Memory (BULK2ARAY)

This module brings the next scanline of data (X:Y component probability values) from the input buffer to the arrays.

Link X:Y data which is currently in columns 64-127 is moved to reference X:Y data area (columns Ø-63).

The next scanline of data is then transferred to arrays at columns 64-127 and processing of the next reference scanline is ready to begin.

INPUTS:

1. Link X:Y values in columns 64-127
2. X:Y component values in the buffer area - 3 scanlines of data per buffer.

PROCESS:

1. X:Y probability value data from columns 64-127 is moved to columns Ø-63.
2. X:Y probability value data from the input buffer is transferred to arrays in columns 64-127.
3. SCANIN counter which counts/down total # of scan lines processed and total # of scanlines per buffer is decremented.

OUTPUTS:

1. Reference X:Y probability value data on array in first 64 Bit slices
2. Link X:Y probability value data in array in second 64 Bit slices
3. Updated SCANIN counter.

CALLED BY: EXEC

SUBROUTINES CALLED: BULK2ARAY

GFUNCTION - Compute G Component of Probability Update Equation

This module computes G(L,K,M) where G(L,K,M) is one half of the equation which determines the correlation value between a reference point and each of its 4 link neighbors.

INPUT:

1. Reference X:Y probability values in columns Ø-63:8 - 8-Bit fields.
2. Link X:Y probability values in columns 64-127:8 8-Bit fields.
3. value of Ø-3 in KCOUNT group counter.
4. value of Ø-3 in LINK link direction.

PROCESS:

1. using KCOUNT as group indicator
   a. SUMX = XR(K) + XL(K)
   b. SUMY = YR(K) + YL(K)

   SUMX and SUMY are preserved for use by the FFUNCTION routine
2. using LINK as link direction values calculate functiong as follows:

a.	link = 3	functiong = SUMX
b.	link = 2	functiong = .7Ø7(-SUMX + SUMY)
c.	link = 1	functiong = SUMY
d.	link = Ø	functiong = .7Ø7(SUMX + SUMY)

OUTPUT:

1. SUMX and SUMY to be used by FFUNCTION routine
2. functiong is scaled 1.9.6

CALLED BY: EXEC

FFUNCTION - Compute F Component of Probability Update Equation

This module computes one-half of the probability equation which determines the correlation value between a reference point and each of its 4 link neighbors.

INPUTS:

1. reference X:Y probability values in columns Ø-63 in 8 8-Bit fields.
2. link X:Y probability values in columns 64-127 in 8 8-Bit fields.
3. value of Ø-3 in KCOUNT which is the group counter.
4. SUMX,SUMY result fields which have been calculated and saved by GFUNCTION routine.

PROCESS:

1. take absolute value of SUMX and SUMY which have been previously calculated in GFUNCTION.
2. DNOMINATOR = SUMX + SUMY
3. using KCOUNT as group indicator:
   a. DIFFX = XR(K) - XL(K) (reference X - link X)
   b. DIFFY = YR(K) -YL(K) (reference Y - link Y)
4. TAKE Absolute value of DIFFX and DIFFY
5. NUMERATOR = DIFFX + DIFFY
6. QUOTF = NUMERATOR ÷ DNOMINATR
7. multiply QUOTF by K^n(fractional constant dependent upon iteration #).
8. FUNCTIONF = 1 - (QUOTF)·K^n

OUTPUT:

1. FUNCTIONF is scaled 1.1.8.

CALLED BY: EXEC

FTNGXFTNF - Compute Product of F and G Components

This module calculates the correlation value by multiplying the FUNCTIONF and FUNCTIONG which have been previously calculated.

For each link direction there are values calculated, of these the largest is chosen as the best correlation.

INPUTS:

1. FUNCTIONF value previously calculated (scaled 1.1.8)
2. FUNCTIONG value previously calculated (scaled 1.9.6)

PROCESS:

1. PLKM1 = FUNCTIONF * FUNCTIONG
   PLKM1 scaled 1.11.4

OUTPUT:

1. current correlation value PLKM

CALLED BY: EXEC

B2ABEST - Bulk to Array Best Correlation Value

This module transfers the current best correlation of "reference X:Y with link X:Y" values which are in a temporary buffer at location E000 - E2FF to the arrays in columns 144-191.

INPUTS:

1. a temporary buffer storing current best correlation values and its corresponding X:Y probability values.

PROCESS:

1. correlation value and X:Y values are accessed and loaded into the arrays.
2. data in the bulk memory is loaded by using LDBYTS routine. Re-ordering is unnecessary since data was not re-ordered prior to being stored in Bulk memory.

OUTPUTS:

1. a correlation value between a reference X:Y and a link X:Y and the corresponding X:Y probability values.

CALLED BY: EXEC

SUBROUTINES CALLED: LDBYTS

UPDATE - Update Probability Values

This module updates the X:Y component probability values for the link direction being processed and then transfers the updated X:Y probability values to a temporary buffer location beginning at E3ØØ. The updated X:Y values are for only 1 link direction at a time. The X:Y values are updated using the X:Y values that correspond to the best correlation values.

The updated X:Y values will replace the current reference data values.

INPUTS:

1. the X:Y component probability values that correspond to the best correlation value are used.

PROCESS:

1. when the final best correlation value is determined (best of 16 calculated), the corresponding X:Y components are used to update X:Y probability values for a link direction.
2. the final best results are transferred from temporary buffer area to the arrays.
3. UPDATEX (LINK) =(BESTXR + BESTXL) /2

   UPDATEY (LINK) =(BESTYR + BESTYL) /2
4. UPDATEX (LINK) and UPDATEY (LINK) are then transferred to a temporary buffer area beginning at E3ØØ.
5. XR3, YR3 updates for LINK = 3 stored E3ØØ - E3FF
   XR2, YR2 updates for LINK = 2 stored E4ØØ - E4FF
   XR1, YR1 updates for LINK = 1 stored E5ØØ - E5FF
   XRØ, YRØ updates for LINK = Ø stored E6ØØ - E6FF

CALLED BY: EXEC

SUBROUTINES CALLED: STBYTS

A2BBEST - Array to Buffer Best Correlation Values

This module transfers the current best correlation of "reference X:Y with link X:Y" values from the arrays to a temporary buffer area at location EØØØ - E2FF.

INPUTS:

1. a correlation value between a reference X:Y and a link X:Y and the corresponding X:Y probability values.

PROCESS:

1. correlation value and X:Y values are accessed and loaded into temporary storage area.

OUTPUTS:

1. a temporary buffer storing current best correlation between a reference line and a link line and the corresponding X:Y probability values.

CALLED BY: EXEC

SUBROUTINES CALLED: STBYTS

MOVE - Move Reference Data For Next Correlation

This module will rotate the reference data in columns Ø-63 left end around by 16 bits. This is necessary to get all 16 possible combinations of reference data with link data for a particular link direction.

INPUTS:

1. reference data is in first 64 bit slices.

PROCESS:

1. XRØ YRØ XR1 YR1 XR2 YR2 XR3 YR3 becomes XR1 YR1 XR2 YR2 XR3 YR3 XRØ YRØ
2. MOVENUM keeps a count of the number of moves done.

OUTPUT:

1. reference data is rotated left end around by 16 Bits and ready to use in calculations.

CALLED BY: EXEC

BEST - Select Best Correlation Values

This module compares previous best correlation values (those now in temporary buffer area) to the current best correlation values (those values just calculated).

INPUTS:

1. current best correlation value (PLKM)
2. corresponding X:Y values of current best
3. previous best correlation value (BESTPLKM)
4. previous best X:Y values

PROCESS:

1. current best correlation value is compared to previous best correlation value, the current best is flagged if larger than previous best.
2. if current best is larger than previous best, the flagged current best is moved to BESTPLKM field. The corresponding reference end link X:Y values are then moved to BESTXR,BESTYR,BESTXL,BESTYL using KCOUNT (group counter) as the index.

OUTPUTS:

1. when completed, the best correlation values and corresponding X:Y pixel values to date are in the array ready to be put in a temporary buffer area.

CALLED BY: EXEC

MOVENEWXY - Move Updated Values From Bulk to Array

This module transfers the updated X:Y probability values from the temporary buffer area to the reference area in the arrays.

INPUTS:

1. a temporary buffer area containing the updated X:Y probability values for all 4 link directions in the following format.

   E3ØØ - E3FF contains all XR3 then YR3 updates
   E4ØØ - E4FF contains all XR2 then YR2 updates
   E5ØØ - E5FF contains all XR1 then YR1 updates
   E6ØØ - E6FF contains all XRØ then YRØ updates

PROCESS:

1. reference X:Y component probability values are replaced by the updated X:Y component values.
2. XR3, YR3 are transferred first followed by XR2,YR2 as a set, then XR1,YR1 and finally XRØ,YRØ.
3. X:Y values are transferred using LDBYTS routine

OUTPUTS:

1. updated reference X:Y component probability values which are ready to go to an output buffer.

CALLED BY: EXEC

SUBROUTINES CALLED: LDBYTS

A2BUPDATE - Array to Buffer Update Values

This module is designed to transfer updated X:Y component values from the arrays (columns Ø-63) to the output buffer area (locations CØØØ or ØØØØ.

Interlock number Ø is set when an output buffer is full and ready to go. When interlock number Ø is set the CDCIO routine is executed to issue the output command.

INPUTS:

1. updated X:Y probability data for the reference point in columns Ø-63. Each of the 4 sets of X:Y values have been updated.
2. OUTBUFPTR references the output buffer location. It is initialized by CDCIO routine.

PROCESS:

1. scanline X:Y values are accessed and stored into the buffer area.
2. OUTBUFPTR is updated.
3. if a buffer is full, interlock number Ø is set and the CDCIO routine is executed. OUTBUFPTR is reinitialized.

CALLED BY: EXEC

SUBROUTINES CALLED: CDCIO

ALIGN - Align Link With Reference Data

This module aligns the link reference data (columns 64-127) to the reference link data (clolums Ø-63). The link data is moved to a scratch area then shifted up/down/none depending on the link direction being worked on.

INPUTS:

1. link data is in columns 64-127
2. reference data is in columns Ø-63.
3. a link direction value in LINK (value from Ø-3).

PROCESS:

1. using the value in LINK, 32 bits of link data are moved to the last 32 bit-slices
2. again using the value in LINK, the moved link data is shifted up/down/none.
3. these aligned data are shifted left by 32 bits, and then steps 1 and 2 are repeated for the remaining 32 bit slices.
4. link = 3 move reference data to scratch area and shift down by 1 word

   link = 2 move link data and then shift down by 1 word

   link = 1 move link data and no shift

   link = Ø move link data and shift up by 1 word
5. all original reference and link values remain undisturbed for later use.

OUTPUTS:

1. shifted link values in columns 192-255 for use in all correlation calculations.

CALLED BY: EXEC

CDCIO - Keep CDC I/O Busy in Background

This module was written to support double buffered I/O.

INPUTS:

1. WHATNEXT is initialized to 1 in the EXEC module.
2. interlock number Ø is set by A2BUPDATE module when an output buffer is loaded and ready to be output.
3. interlock number 1 is reset by EXEC after an input buffer is exhausted,and CDCIO signals another is ready by setting it.

PROCESS:

The routine repeatedly cycles through four I/O steps. It has two ways of returning - it always returns to the instruction immediately following the call except when it is waiting for A2BUPDATE to set interlock number Ø so that it can continue by issuing an output command.

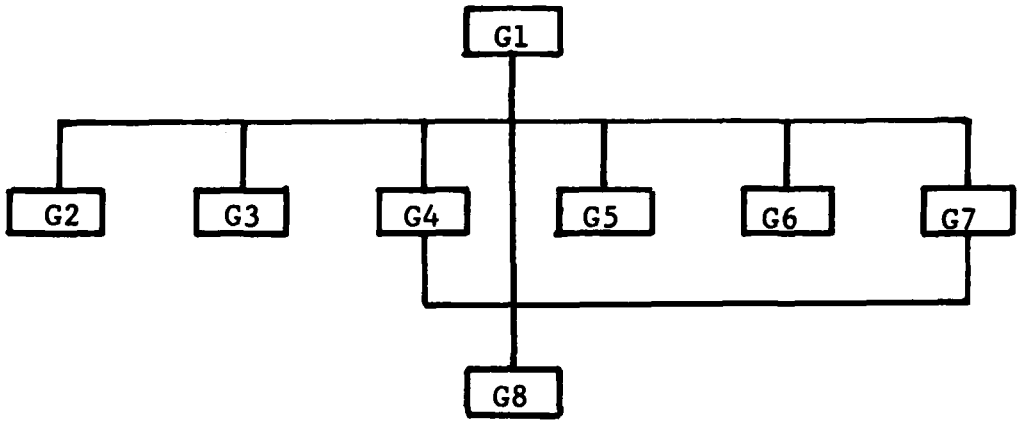
1. Exit unless input is available. When input has been received, continue with the next step in the I/O process, i.e., output in step 2.
2. after step 1 receives input a wait for interlock number Ø to be set by A2BUPDATE is executed and an exit made. When interlock number Ø becomes set, it is reset, and an output command executed. WHATNEXT is bumped to the next step and an exit from the routine made.
3. exit unless the output has completed. When the output request issued in step 2 has completed, the output buffer address is flip-flopped, WHATNEXT is bumped to the next step, a command to input a buffer of data is executed, WHATNEXT is bumped to the next step, and an exit from the routine made. Step 1 will be executed the next time this routine is called.

OUTPUTS:

1. OUTBUFPTR is set to reference a buffer of output data whenever a new one is received.

CALLED BY: EXEC

G. STARAN DISPLAYABLE IMAGE ROUTINES



Page	Program Module
G1	DSPLAYEXC - Main Module For Display Routine
G2	INITDSPLY - Initialize Display Routine
G3	MAXIMUMR - Determine the Maximum R Value
G4	LOADNXCOL - Load Next Column
G5	GTTHRSHLD - Greater Than Threshold
G6	CALCULATR - Calculate R Value From X and Y
G7	SCAN2BULK - Transfer R,X, and Y Pixels From Array to Bulk
G8	IMAGIO - Keep CDC I/O Busy In Background

DSPLAYEXC - Main Module For Display Routines

The DSPLAYEXC is the controlling module for the display calculations.

INPUTS:

1. x:y component pixel values
   There are 4 pairs of x:y pixel values with each pair referencing a link direction.

PROCESS:

1. many initialization functions are performed when INITDSPLY module is executed.
2. to keep I/O busy in the background mode,IMAGIO routine is executed between all calls to major subroutines.
3. LOADNXCOL is executed to transfer a scan line of x:y component probability data from the input buffer area to array memory.
4. display values (RØ,R1,R2,R3) are calculated using the 4 sets of x:y probability values when CALCULATR is executed.
5. MAXIMUMR is executed to choose the maximum R value (x:y components corresponging to MAXR are also kept)
6. GTTHRSHLD is executed to determine if Maximum R is larger than the threshold display value. If not, it is given a pixel value of zero.
7. a scanline of pixel data is transferred from the array to an output buffer by executing SCAN2BULK
8. the last output contains only 2 scan lines - it is output by DSPLAYEXC.

OUTPUT:

1. three display images R values (weighted x:y)
   X values
   Y values

CALLED BY: EXEC

INITDSPLY - Initialize Display Routines

This module performs the initializations for the routines that produce a displayable image from the x:y component probability values.

INPUTS:

uninitialized counters, flags, I/O instructions, etc.

PROCESS:

1. many initializations are done
   a) Input buffer pointers are inited
   b) Output buffer pointers are inited
   c) I/O control handler is inited to 21
   d) the number of Input buffers is inited

OUTPUTS:

1. All necessary initializations needed for display routines.

CALLED BY: DSPLAXEXC

MAXIMUMR Determine the Maximum R. Value

This module will determine the maximum R value of the 4 calculated.

The corresponding x:y probability values used to calculate the maximum R are also saved.

INPUTS:

1. Four display R values RØ, Rl, R2, R3
2. The corresponding x:y values for this scanline XRØ, YRØ, XRl, XR2, YR2, XR3, YR3

PROCESS:

1. RØ is moved to MAXR maximum R. XRØ YRØ are moved to MAXX, MAXY respectively.
2. The next R value is compared to the current MAXR. If any of the next remaining R values are greater than the current MAXR, that R value and its x:y values are moved into MAXR, MAXX, MAXY.
3. Step 2 is repeated for the next 2 R values.

OUTPUT:

1. The maximum R display value and the x:y component probability values, MAXR, MAXX, MAXY. These are the values that will be displayed.

CALLED BY: DSPLAYEXC

LOADNXCOL - Load Next Column

This module transfers a scan line of probability data (x:y component values) from an input buffer area to array memory.

INPUTS:

1. Probability data (x:y component values) in a buffer area.
2. SCANIN is the count of remaining scan lines in an input buffer.
3. WHICHINPB indicates input buffer status.

PROCESS:

If the current input buffer is empty (SCANIN=0), the status of the next input buffer is examined. If not full yet, IMIGIO is called until it is full. SCANIN is re-inited to 3 scan lines per input buffer, and BULK2ARAY is called to load the next scan line. If SCANIN is not zero, LOADNXCOL is exited. Otherwise, the status of the current input buffer is set to empty and WHICHINPB is cycled to select the next input buffer. IMIGIO is called in case it was waiting for an empty input buffer. INBUFFTR is inited to the beginning of the new input buffer area and LOADNXCOL is exited.

OUTPUTS:

A scan line of probability data moved from bulk to array memory. SCANIN, WHICHINPB, and INBUFFTR are maintained.

CALLED BY: DSPLAXEXC

SUBROUTINES CALLED: BULK2ARAY
IMIGIO

GTTHRSHLD - Greater Than Threshold

This module checks if the maximum R value (MAXR) is larger than the desired threshold value.

INPUTS:

1. Maximum display value R and its component x:y probability values, MAXR,MAXX,MAXY

PROCESS:

1. MAXR values which are greater than or equal to the threshold value are flagged.
2. Those MAXR values not flagged are set to a zero pixel value. The corresponding MAXX and MAXY values are also set to the zero pixel value.
3. the sign bit of MAXX and MAXY is complimented to get display pixel values.

OUTPUTS:

1. MAXR, MAXX, MAXY display values which are ready to be displayed.

CALLED BY: DSPLAYEXC

CALCULATR - Calculate R Value From X and Y

This module calculates four R values where R is a displayable pixel value that is a combination of the X:Y component probability values, $R = \sqrt{X^2 + Y^2}$.

R is used to determine which edge correlation value is to be displayed.

INPUTS:

1. X:Y probability values for reference scanline.

PROCESS:

1. For each pair of X:Y probability values, R is calculated.
2. $R\emptyset = \sqrt{XR\emptyset^2 + YR\emptyset^2}$
   $R1 = \sqrt{XR1^2 + YR1^2}$
   $R2 = \sqrt{XR2^2 + YR2^2}$
   $R3 = \sqrt{XR3^2 + YR3^2}$

OUTPUTS:

1. Four R display values from which the maximum will be chosen

CALLED BY: DSPLAYEXC

SCAN2BULK - Transfer R,X, and Y Pixels From Array to Bulk

This module transfers three scan lines of pixel data from the array to an output buffer.

INPUTS:

1. Maximum calculated R pixel value and its corresponding X:Y component pixel values.
2. Output buffer pointers: OUTBUFR, OUTBUFX, OUTBUFY
3. SCANOUT, the counter for 30 scan lines per buffer

PROCESS:

1. MAXR value is aligned on a field multiple of 8 for transfer to bulk memory.
2. MAXR values are transferred to bulk area 9ØØØ,and corresponding MAXX transferred to bulk area AØØØ, and MAXY transferred to bulk area BØØØ.
3. SCANOUT counter is counted down
4. If SCANOUT is zero, it is re-inited to 30, and IMIGIO is called until it is waiting on an output buffer to be filled. ILOCK #0 is then set to indicate that an output buffer is full and IMIGIO is called again to output the newly filled buffers.

OUTPUTS:

1. MAXR, MAXX, MAXY display values in 3 separate buffer areas.

CALLED BY: DSPLAYEXC

SUBROUTINES CALLED: OUTBYTES
IMIGIO

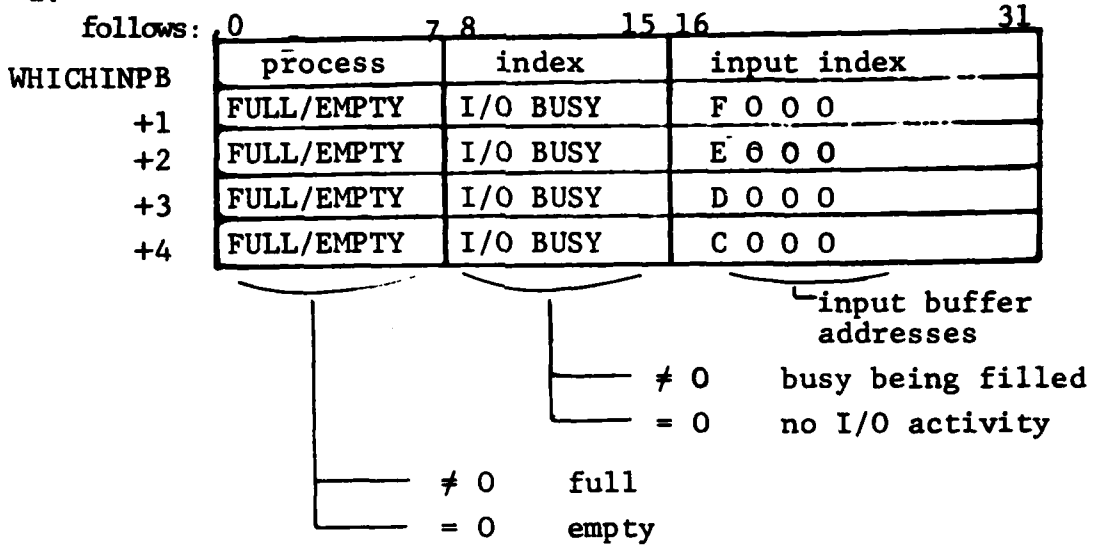
IMAG"O - Keep CDC I/O Busy In Background

This module supports the Image generating process in background mode.

There are 4 input and 3 output buffers.

INPUTS:

1. WHICHINPB is a five word control structure formatted as follows:



The lower 16 bits of WHICHINPB (bits 16-31) indexes the next input buffer area to receive data. The upper 16 bits of WHICHINPB (bits 0-15) indexes the next input buffer to process. The next input buffer cannot receive data unless it is empty. Similary the next input buffer cannot be accessed for data to process until the full flag is set.

2. NEXTIO2DO is initialized to 21 in the INITDSPLY module. The first of 10 input buffers has been requested by INITDSPLY.
3. NUMINBUFS is the number of input buffers to receive.

PROCESS:

This routine is called by the DSPLAXEXC between all major subroutine calls in order to keep CDC I/O as busy as possible. It repeatedly cycles thru a control table of I/O functions until 169 input buffers have been received from the CDC. For every ten buffers

of input data (probability values @ 3 scan lines per buffer), there are three buffers of output data (each buffer contains 30 scan lines of 8 bit image pixels). The three buffers of output data represent the R,X, and Y images, i.e., different components of the same image, $R = \sqrt{X^2 + Y^2}$.

OUTPUTS:

1. WHICHINPB input index is cycled after completion of an input.
2. NEXTIO2DO is cycled whenever an I/O function has completed.
3. NUMINBUFS is decremented whenever an input is executed.

CALLED BY: DSPLAYEXC
SCAN2BULK
LOADNXCOL

H. PROGRAM USAGE

The entry and exit conditions are described for a main program that uses the relaxation iteration edge detection FORTRAN subroutines.

A call to LDSTAR that looks like

```
CALL LDSTAR(10HRELAX␣ALD␣,10001B,IACTION,ISTAT)
```

must be the first subroutine call made in order to load and execute the relaxation software in the STARAN. This subroutine must also be called after a terminate signal is sent to the STARAN, and the user wishes to continue executing the relaxation software under a different set of conditions.

1. RELAX Entry Conditions

This subroutine, like LDSTAR, is called only once during a session for a particular set of conditions. It is the second STARAN related subroutine that must be called. A call to RELAX looks like

```
CALL RELAX(FILTER,NORMLZ,ITBDSP,ITHRSH)
```

The subroutine arguments are defined as follows:

a) FILTER(201) is a real array whose first value must be equal to 3, the dimension of the four filters that follow. The filters are used during convolution processing to compute the probability that edges exist in the eight compass directions: filter #1 detects edges along a -45^{o} azimuth, filter #2 detects edges along a 0^{o} azimuth, filter #3 detects edges along a 45^{o} azimuth, and filter #4 detects edges along a 90^{o} azimuth. The filters are in row order and follow one another in the FILTER array. The value of the largest filter element must be in the range

$$|f_n(i,j) - avg_n| < 128$$

where $f_n(i,j)$ is the largest filter element of the n^{th} filter and avg_n is the average value of all elements for the n^{th} filter.

b) NORMLZ is a normalization value and must be in the range $0 \leq NORMLZ \leq 255$. The result of a convolution is normalized by dividing by the following product:

$$\sigma_{f_n} \bullet \sigma_X$$

where σ_{f_n} is the standard deviation of the n^{th} filter and σ_X is the standard deviation of the 3x3 neighborhood of the image wherein the convolution is calculated. The σ_X value is dependent upon the pixel values of each 3x3 neighborhood. The NORMLZ value provides the user with a fixed normalization value over the entire image.

c) ITBDSP is the number of iterations between displays and must be $0 \leq ITBDSP \leq 255$. It will be set to one if initially less than or equal to zero.

d) ITHRSH is a threshold value used during the calculation of the displayable image pixels from the probability data. Each pixel is represented by 4 pairs of X,Y components of the probability that an edge exists in each of the four directions. The largest X,Y pair is selected based upon

$$R = \sqrt{X^2 + Y^2}$$

where R is the largest value. If R<ITHRSH, it and its associated X and Y component values are set to zero.

A named COMMON is used to provide DIAL image information to the RELAX and ITRATE subroutines and must look like

```
      COMMON/INFO/IMAGNM(4),IPROBS(4),NAME(4),NAMR(4),NAMX(4),
     1            NAMY(4),LABEL(2602),LABELP(2602)
```

The definition of each item is as follows:

a) IMAGNM(4) is the name of the input source image and must be a 512^2 x 8 bit pixel image

b) IPROBS(4) is the name of the probability data "image" produced by the STARAN. It must consist of 171 scan lines, each scan line has 1640 pixels, and each pixel is 60 bits.

c) NAME(4) is the name of an image identical to IPROBS, and is used by the ITRATE subroutine to accumulate updated probability values derived from the data in the IPROBS image.

d) NAMR(4),NAMX(4),NAMY(4) are the displayable 512^2 x 8 bit pixel images derived from the probability data in the IPROBS image.

e) LABEL(2602) is the label header descriptor of the 512^2 x 8 bit pixel images: IMAGNM,NAMR,NAMX, and NAMY.

f) LABELP(2602) is the label header descriptor of the 171x1640x60 bit pixel probability data images IPROBS, and NAME.

The DIAL status of each of the images in the COMMON upon entry to RELAX is:

a) IMAGNM(4) requires a FIND before it can be DREAD.
b) IPROBS(4) is ready to be DWRITE'n.
c) NAME(4) must not exist.
d) NAMR(4),NAMX(4), and NAMY(4) are ready to be DWRITE'n.

The DIAL status of each of the images in the COMMON upon return from RELAX is:

a) IMAGNM(4) has been completely DREAD.
b) IPROBS(4) is ready to be DWRITE'n.
c) NAME(4) does not exist.
d) NAMR(4),NAMX(4), and NAMY(4) have been completely DWRITE'n and then DCLOSE'd.

2. ITRATE Entry Conditions

This subroutine is repeatedly called until the user no longer wants to perform iterations upon the probability estimates. A call looks like

CALL ITRATE(ITBDSP)

where ITBDSP must be either identical to the ITBDSP value when RELAX was called; or if equal to minus one, the STARAN will terminate processing. If ITBDSP is not equal to the original value when RELAX was called, an I/O error will occur.

The named COMMON described for the RELAX subroutine, and the DIAL status of the images defined by the COMMON must be identical to the entry status described for the RELAX subroutine. The return status of each of the images is also identical to that described for RELAX. A possible exception is that the IMAGNM(4) image is not used by the ITRATE subroutine.

I. PROGRAM MAINTENANCE

The source cards for all of the relaxation software is contained in two drawers labeled STARAN EDGE DETECTION PROBABILITY and STARAN EDGE DETECTION ITERATION in the room adjacent to the keypunch room at ETL.

1. CDC6400 FORTRAN Subroutines

The FORTRAN subroutines are organized into a library of subroutines called RELXSTR. There are three sets of job control cards: one is used to create an entirely new library, and another is used to simply update the library by recompiling one or more subroutines. The third is used to re-link the modified library with the main calling program, EDGERLX, for execution on the DIAL system.

The job control stream used to generate a complete new RELXSTR library is:

```
ETRWL.  COMPILE RELXSTR LIBRARY
TASK,TNET76060,PWETLFB,TRTS.  GENERATE A NEW LIBRARY
PURGE(A,RELXSTR,ID=ET71333)
REQUEST,NEWLIB,*PF.
FTN(A,B=RELXSTR,BL,C=0,EL=F,R=3,OPT=2)
EDITLIB(USER)
CATALOG(NEWLIB,RELXSTR,ID=ET71333)
CLIST(T=U)
#                                   A 7/8/9 MULTIPUNCH CARD
{LDSTAR SOURCE DECK}
{RELAX SOURCE DECK}
{SENDIM SOURCE DECK}
{GTPROB SOURCE DECK}
{ITRATE SOURCE DECK}
#                                   A 7/8/9 MULTIPUNCH CARD
REWIND(LGO)
LIBRARY(NEWLIB,NEW)
ADD(*,LGO)
FINISH.
ENDRUN.
#                                   A 6/7/8/9 MULTIPUNCH CARD
```

The job control stream used to recompile one or more subroutines in the RELXSTR library is:

```
ETRWL.  RECOMPILE SOME RELXSTR SUBROUTINES
TASK,TNET76060,PWETLFB,TRTS.  MODIFY RELXSTR SUBROUTINE(S)
ATTACH(NEWLIB,RELXSTR,ID=ET71333)
FTN(A,BL,C=0,EL=F,R=3,OPT=2)
EDITLIB(USER)
EXTEND(NEWLIB)
CLIST(T=U)
#                                   A 7/8/9 MULTIPUNCH CARD

  { ONE OR MORE SOURCE DECKS }

#                                   A 7/8/9 MULTIPUNCH CARD
REWIND(LGO)
LIBRARY(NEWLIB,OLD)
REPLACE(*,LGO)
FINISH.
ENDRUN.
#                                   A 7/8/9 MULTIPUNCH CARD
#                                   A 6/7/8/9 MULTIPUNCH CARD
```

The job control deck required to re-link a new or modified RELXSTR library with the main program EDGERLX for execution on the DIAL system is:

```
BROWN.    CREAT NEW DIAL PROGRAM MODULE AFTER
TASK(TNET75454,PWETCSL,TRTS) RECOMPILING
ATTACH(CLIST,ID=ET71333)
ATTACH(RELXSTR,ID=ET71333)
ATTACH(STNCOM,ID=ET71345)
ATTACH(Q9OBJCT,ID=ET71333)
ATTACH(LGO,EDGERLY,ID=ET75454)
REWIND(LGO)
LDSET(LIB=Q9OBJCT/RELXSTR)
LOAD(LGO)
LOAD(STNCOM)
NOGO(PMBIN)
ATTACH(Q9IMAGE,ID=ET71333,PW=XR,RW=1)
EDITLIB(USER)
EXTEND(Q9IMAGE)
CLIST(T=U)
EXIT(S)
CLIST(T=U)
#                                   7/8/9 MULTIPUNCH CARD
REWIND(PMBIN)
LIBRARY(Q9IMAGE,OLD)
REPLACE(*,PMBIN,FL=0,AL=1,FLO=0)
FINISH.
ENDRUN.
#                                   6/7/8/9 MULTIPINCH CARD
```

Whenever any subroutine in the RELXSTR library is modified, or any subroutine related to EDGERLX (including EDGERLX) is modified, this job control stream must be executed in order to create a new DIAL program module.

2. STARAN Routines

The STARAN routines are divided into two separate modules on the user's disc at ETL: 1) the initial probability computation software in UIC[40,33], and (2) the iteration and displayable image software in UIC[40,35]. All source decks are bounded by the job control cards required for reassembly. Place the deck(s) to be reassembled into the card reader and type

```
BA    CR:↵
```

on the DECWRITER. After reassemblies have been performed, type

```
BA  DK1:LINKP[40,33]↵
```

to generate a new load module for the initial probability computation software, or type

```
BA  DK1:LINKI[40,35]↵
```

to generate a new load module for the iteration and displayable image software.


DA
FIL