JPL PUBLICATION 79-79

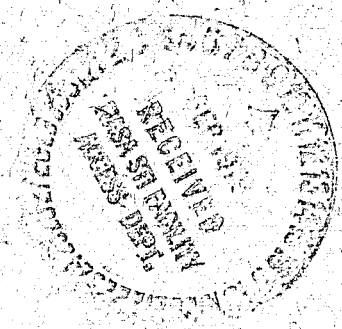
Low-Level Processing for Real-Time Image Analysis

R. Eskenazi
J. M. Wilf

September 1, 1979

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California



JPL PUBLICATION 79-79

Low-Level Processing for Real-Time Image Analysis

R. Eskenazi
J. M. Wilf

September 1, 1979

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

ABSTRACT

This paper describes a system that detects object outlines in television images in real-time. A high-speed pipeline processor transforms the raw image into an edge map and a microprocessor, which is integrated into the system, clusters the edges and represents them as chain codes. Image statistics, useful for higher level tasks such as pattern recognition, are computed by the microprocessor. Peak intensity and peak gradient values are extracted within a programmable window and are used for iris and focus control.

The algorithms implemented in hardware and the pipeline processor architecture are described. The strategy for partitioning functions in the pipeline was chosen to make the implementation modular. The microprocessor interface allows flexible and adaptive control of the feature extraction process.

The software algorithms for clustering edge segments, creating chain codes, and computing image statistics are also discussed. Finally, a strategy for real-time image analysis that uses this system is given.

CONTENTS

Figures

I. INTRODUCTION

A vision system that recognizes objects in a scene, determines their locations, and tracks their movements, can be partitioned into two layers. The first, the low-level processor, transforms the raw image into a compact feature representation. The second layer, the high-level processor, performs scene analysis using these features and a priori knowledge of the scene. The features considered in this paper are complete object outlines and their characteristic statistics (perimeter, area, center of mass, etc.).

When the scene varies with time and objects must be followed, or visual feedback given to mechanical effectors, feature extraction must take place in real-time. The definition of "real-time" will depend on how quickly the scene is changing. For this discussion it will correspond to changes detectable when several frames per second are sensed. Even with the fastest general-purpose computer available today, feature extraction will take at least a few seconds for each image frame. Therefore, special purpose hardware must be used if features are needed in real-time.

A real-time low-level processor that extracts edge information at the video rate of thirty frames per second has been implemented. The system is flexible. That is, portions of the logic are programmable, control parameters can be adjusted dynamically by the high-level processor, and objects of interest can be framed in a window. The window, which can range from a single pixel to the entire image, allows data to be transmitted only from the image area of interest. The system is adaptive, automatically adjusting the camera's focus and iris to reflect changes in the scene. Finally, the system has a modular hardware implementation.

II. ARCHITECTURE

Most low-level image processing algorithms (i.e., enhancement, edge detection, thinning, etc.), are built from local operators that require access to a small number of picture elements at any one time. These operators, usually requiring simple arithmetic operations, are computed for every pixel in the image, and are called computational units (CUs) when they are implemented in hardware for fast processing. The question is how to arrange the CUs so that algorithms are applied to an entire image in the least amount of time.

One approach, studied by Duff [1] and Dyer, et al. [2], arranges identical CUs in an array that processes the entire image in parallel. In this scheme, the execution time for an algorithm would be the execution time of one CU. Even using a highly complex operator, an array of CUs will transform an entire image in a few microseconds. Unfortunately, there are practical drawbacks to this architecture. Parallel arrays require as many CUs as there are pixels in the image--over a quarter of a million for a typical 512 x 512 image. The total number of connections needed (the number of pixels accessed by each CU times the number

of CUs), is an even more serious problem [3]. One way to eliminate the connection problem is to integrate the CUs and the sensor on one circuit. However, to justify the parallel processing, the IC should also contain all the processing necessary to reduce the data to a few features representing the sensed image. This amount of logic cannot fit on present-day, two-dimensional solid state devices. Finally, since present-day sensors output serial data, a parallel array processor, separated from the sensor, would have to remain idle during the entire time the image is being sensed. For these reasons, the parallel array approach was not used in this system.

The converse of the parallel array approach is to scan the entire image with a single CU. First, the raster output of the camera must be converted to a window that contains all the points the CU needs to access in parallel. Figure 1 shows raster-to-window conversion (R/W)--how serial image data are converted to the 3 x 3 window used by the CU. Input pixels are stored in a pair of line-long shift registers, giving parallel access to a 3 x 3 window. The CU must be ready to accept a new 3 x 3 window every time a new pixel is received, forcing the CU to execute in no more than the pixel sampling time. The CU's execution time, unimportant in parallel array processing, is critical in this scheme.

Timing problems, caused by large propagation delays within a CU, are overcome by breaking the CU into several computational subunits (CSUs). These CSUs, separated by latches that are simultaneously clocked at the pixel sample rate, form the pipeline processor shown in Fig. 2. The total propagation delay, which must be kept less than the sampling time, determines the amount of logic in each CSU. This strategy has the limitation that data lines cannot be fed back from the output of a CSU to the input of a previous one. Functions that require feedback must be implemented in a single CSU. Such CSUs may need faster logic circuit components to keep execution below the pipeline clocking time.

Thus far, only one processing step has been considered. However, complex multi-step processes can be performed by combining raster to

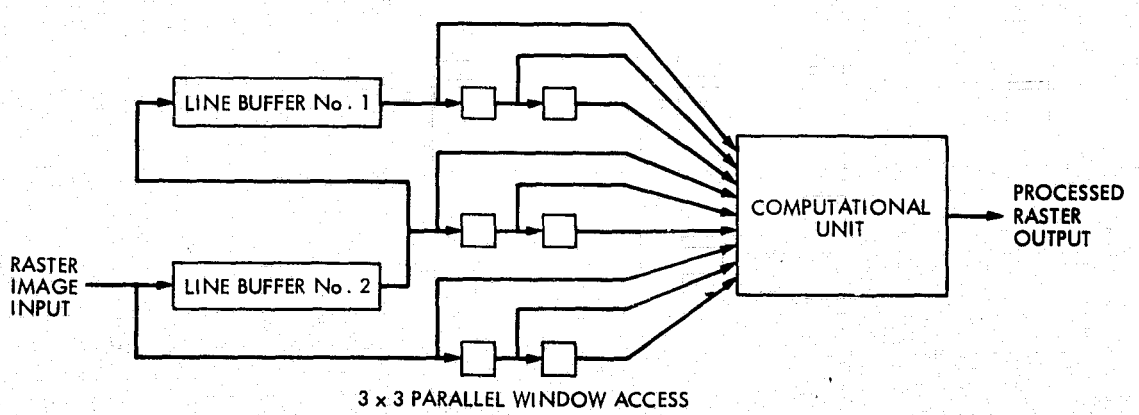


Figure 1. Raster-to-Window Conversion Provides Parallel Access to a 3 x 3 Neighborhood for a Computational Unit.

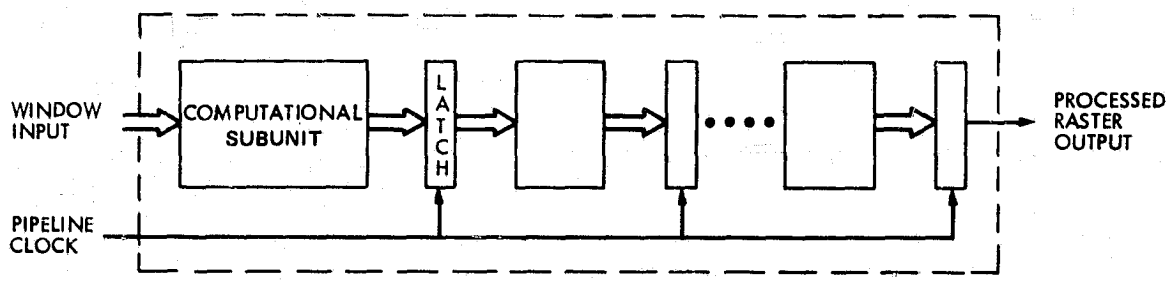


Figure 2. Partitioning of a Computational Unit into Computational Subunits.

window converters and CUs in a pipeline, as shown in Fig. 3. The total delay between the raw image and the processed output is the number of lines stored in the raster-to-window converters plus the total number of CSUs in the system.

III. SEQUENTIAL PROCESSING FOR EDGE DETECTION

Using the pipeline architecture discussed above, a three-step operation that produces partial object outlines from a raw image has been implemented. The CUs that perform each step are an edge enhancer, an edge detector, and an edge processor.

The edge enhancer, Fig. 4, the first CU in the pipeline, differentiates the raw image (digitized to eight bits) by applying a weighted filter. The result is a gradient magnitude image, having a ten-bit value for each pixel that represents how sharply the gray level changes. This edge enhancer was chosen because it combines good performance with simple hardware implementation [4]. The gradient magnitude, G(i,j), is

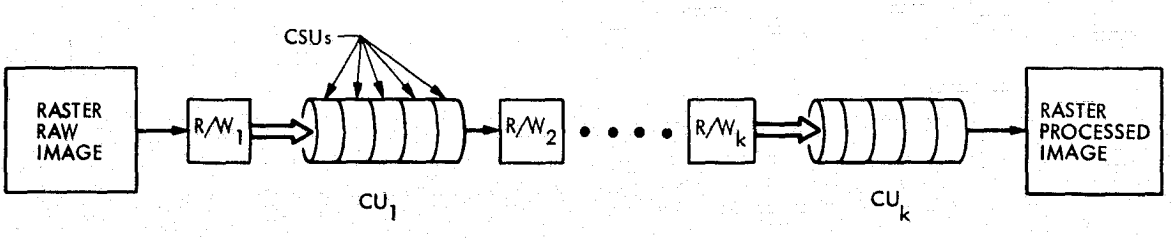


Figure 3. Pipeline Image Processor Formed by Cascading Several Computational Units.

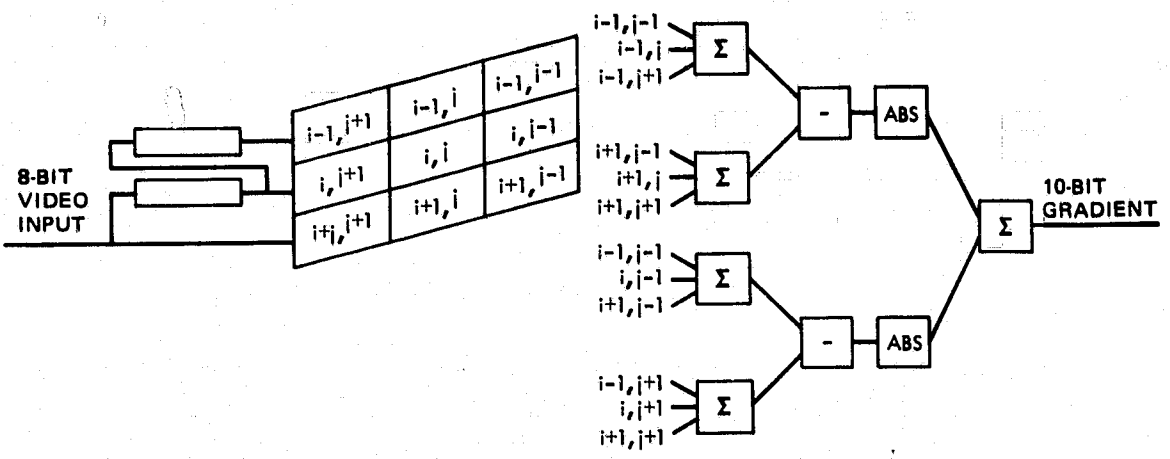


Figure 4. Block Diagram of the Edge Enhancer.

computed at each pixel, whose gray scale value is P(i,j), as the following function of the pixel's eight nearest neighbors:

$$G(i,j) = |P(i-1,j-1) + P(i,j-1) + P(i+1,j-1) - P(i-1,j+1) - P(i,j+1) - P(i+1,j+1)| + |P(i-1,j-1) + P(i-1,j) + P(i-1,j+1) - P(i+1,j-1) - P(i+1,j) - P(i+1,j+1)|$$

An edge detector, whose output is a binary edge map, is the second CU. Like enhancement, edge detection is a type of weighted filtering. The inputs of this filter are the gradient magnitudes, G(i,j)s, defined above. S(i,j), the value of the filter, is defined as follows:

$$S(i,j) = \sum_{k=-1}^{1} \sum_{\ell=-1}^{1} W_{k\ell} * G(i+k, j+\ell)$$

where the $W_{k\ell}$ are programmable integer weights and W00 is minus the sum of the other weights. Each point in the binary edge image, E(k,j), is set to one (edge) if S(i,j) is positive, and to zero (no edge) otherwise.

Once an edge map is obtained, single-point elimination, edge thinning, single-pixel gap filling, and generation of a list of node and end points may be desired. All this is done by the third CU, the edge processor. The edge processor makes decisions for each pixel by checking the states of its eight nearest neighbors. Since each neighbor may only be in one of two states in the binary edge map, there are 2 ** 8, or 256, possible conditions for every pixel. The 256 possibilities

define an address into a two-bit-wide lookup table. The lookup table is programmable for flexible decision making.

Two special registers are used for windowing. The first contains the upper left-hand corner (ULC) and the second contains the lower right-hand corner (LRC) of the window. Both ULC and LRC are programmed by the microprocessor. Using an image location address counter, the hardware keeps track of the raster scan. The hardware compares the scan address to the window address registers and generates a logic signal to indicate the area of the image lying within the window. When the scan reaches the window's LRC, an interrupt is generated for the microprocessor.

Two identical CUs are enabled within the programmable window to detect peak-image intensities. The first, having direct access to the raw video, computes the maximum video level within the window. The second, attached to the gradient output of the first CU, computes the peak gradient value within the window.

IV. IMPLEMENTATION

The input devices to the system are GE TN-2000 solid state television cameras with a resolution of 188 x 244 elements. The camera's output is an EIA RS-170 format signal of thirty frames per second and the system is able to lock onto any standard input signal of this form. The pixel sampling rate for the GE cameras is 3.58 MHz, making the pipeline clocking time 280 nsec.

All logic in the system is implemented with TTL 74LS series circuits. Computational units are designed to handle clocking times as low as 150 nsec. By replacing the 74LS with 74S components, the clocking time can be lowered to 90 nsec, more than adequate for camera resolutions of 512 x 512.

The raster-to-window converters are built with fast bipolar RAMs (35 nsec access time) instead of shift registers to accommodate different line resolutions that may be needed in the future.

The microprocessor used in the system is a DEC LSI 11/03. A DMA controller with programmed I/O capability sends and receives data to or from the pipeline processor through a bidirectional bus (Fig. 5). After the third CU is applied, processed edges are stored in the computer memory via DMA for software processing. During DMA transfer of the edge map, only about 10% of the CPU time is occupied, leaving the CPU about 90% for running software. When a small window is set, the overall duty cycle of the CPU is higher since it has 100% of the time while the non-windowed area of the image is being scanned.

Camera lenses are controlled by velocity servos. Digital velocity values, dictated by the microprocessor, are converted to analog signals and fed to the servo controllers.

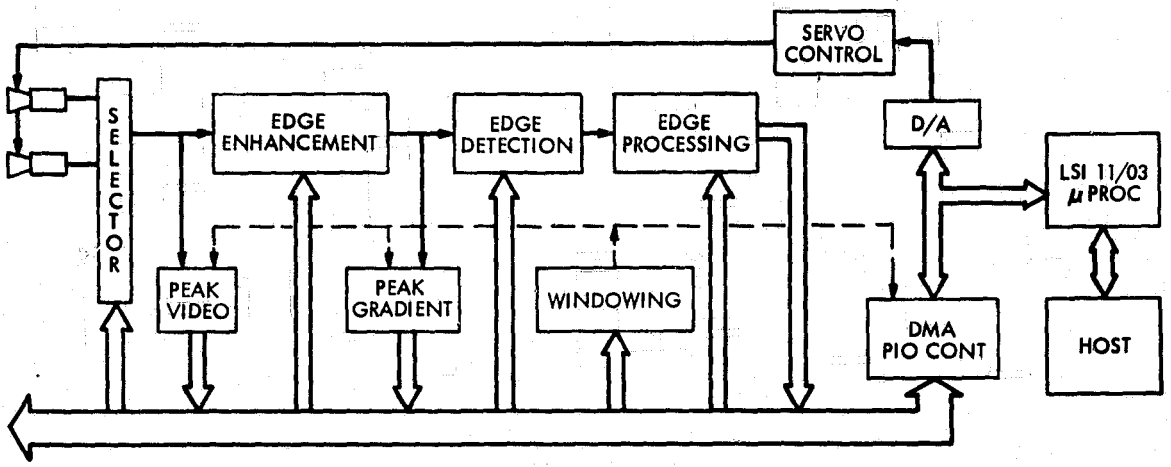


Figure 5. System Configuration of the Real-Time Image Feature Extractor.

A special channel in the video input selector is dedicated for switching between a pair of stereo cameras on alternate fields. Since the pipeline operates on one field at a time, switching cameras lowers the vertical resolution by a factor of two, but enables the system to make three-dimensional measurements in frame time.

V. SOFTWARE FOR LOW-LEVEL PROCESSING

By "software for low-level processing" we mean the programs resident on the microprocessor. These programs adjust the pipeline's programmable control parameters, giving the system flexibility and enabling it to adapt to a scene. Software must control the camera's focal length and iris diameter, have a predominant role in edge clustering, extract chain codes, and calculate image statistics. Real-time speed requires that the software be optimized for minimum execution time, implying that it be written in fast assembly language modules. System adaptability to the scene implies that the software be event driven using an interrupt structure. System programming considerations, such as communication with the high-level processor, are beyond the scope of this discussion.

The low-level processor must control the cameras to ensure that the proper amount of light is received and that objects of interest remain in focus. All focusing and irising are done inside the programmable window set by the high-level processor around the objects of interest. This allows the system to get the proper amount of light, even if brighter objects are in the scene. Similarly, the cameras can be focused on a particular object without being affected by other objects located at different depths.

Most researchers agree that clustering cannot be done solely by a low-level process [5]. The problem, then, is to find a clustering strategy that minimizes dependence on the high-level processor, and thus execution time.

Assume that the clustering program is given a list of the end and node points of all the edge segments in the binary edge map. This list, as pointed out above, is found during the same process that thins the edge map. The strategy will be to let the low-level processor do a global "best guess" clustering at the first pass and let the higher level processor make local corrections as more information becomes available or as the scene changes.

One choice of a best guess global clustering is an envelope operation that joins the edge segments into a silhouetted image. Each endpoint is connected to the closest neighbor such that the connecting segment does not intersect any other segment. The primary limitation of this method is that extraneous edges will distort an object's envelope. By setting the initial edge detection threshold to a high value, most extraneous edges can be eliminated at the cost of a few more gaps to fill.

Chain codes are an ordered list of numbers that represent the orientation of line segments along the boundary curve. These codes have been extensively investigated by Freeman [6], and provide a succinct and convenient representation of object boundaries. Many algorithms, such as corner detection [7], contour following, and image statistic calculation [8] run more quickly using chain codes. The data compression of a chain-coded edge can speed up communication time between the low- and high-level processors.

These advantages must be weighed against the major disadvantage of chain codes — the time it takes to extract them. Several methods aimed at speeding up the chain encoding process have been proposed. Yakimovsky [9] has proposed a one-pass raster scan technique; Wong [3] has described a method that works from the list of node and boundary points; and, Sobel [10] has given a structure that allows hardware to aid chain code extraction. All these techniques will chain code the entire edge map. If the chain codes are stored as a linked list, then the list can be locally updated, rapidly adding or deleting chain-coded segments as the scene changes.

The perimeter, area, and arbritrary moments are calculated from a region's boundary. Combinations of these statistics describe the location, orientation, and shape of objects in space. Wong and Hall [11] show how certain linear combinations of moments, the invarient moments, are used for object recognition. Wilf and Cunningham [8] derive the formulas for calculating arbitrary moments and discuss their computational properties.

One useful property of region moments is their linearity. Let R be a region composed of two sub-regions R_1 and R_2, as shown in Fig. 6. Let M(R) signify the moments of R. Then, $M(R) = M(R_1) + M(R_2)$. The perimeter of R, P(R), is easily computed from R_1, R_2, and the length of the common edge, C, between R_1 and R_2, L(C): $P(R) = P(R_1) + P(R_2) - 2 * L(C)$. This property implies that once statistics have been calculated for the entire scene, local adjustments can be made without having to do a major global recalculation.

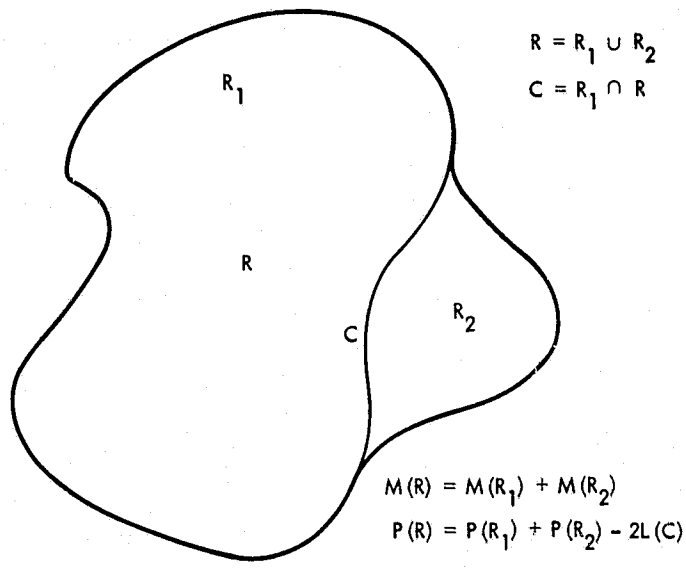


Figure 6. Two Regions With Common Boundary Segments.

The pattern that has emerged from the analysis of clustering, chain coding, and statistics calculation is that these processes are performed only one time globally. Once a good initial guess image boundary is created, all adjustments and recalculations are done locally, involving only the altered portion of the image. This gives us the following scenario for real-time scene analysis:

1) Using an initial window that includes the entire scene, the cameras are adjusted for proper light and focus.

2) The binary edge map and a list of segment end points and node points, created by the pipeline, are read into the microprocessor's memory.

3) Edge gaps are closed to form object silhouettes. Each object boundary is chain coded and its perimeter, area, and low-level moments are calculated. This information is then passed to the high-level processor for analysis.

4) The higher level processor identifies a region of interest and sets a window around it.

5) The threshold is lowered inside the window to get more accurate edge information.

6) Focus and iris are adjusted, using the new window to center on the object of interest.

7) The lower threshold reveals new edges in the edge map and features inside the object. The chain code representation is refined, image statistics are readjusted, and the new information is passed back to the higher level processor.

8) The higher level processor, tracking the changes in the object's boundary and statistics, requests a new window and predicts new edges on the basis of a priori models of the scene.

9) Steps (5) through (8) are repeated until the scene has been sufficiently analyzed.

VI. CONCLUSIONS

A pipeline architecture is suitable for high-speed sequential processing of images. In the low-level processor a raster-to-window converter, along with a computational unit, form a generic building block. The usage of uniform, pipelined building blocks makes the implementation modular. By chain code representation the system reduces the amount of data in an average image by about two orders of magnitude. This reduction and representation enables high-level algorithms to operate on images in near real-time.

REFERENCES

[1] M. Duff, "CLIP4, A Large Scale Integrated Circuit Array Parallel Processor," Proceedings of the International Conference on Pattern Recognition, Coronado, California, November 1976.

[2] C. Dyer, and A. Rosenfeld, "Cellular Pyramids for Image Analysis," University of Maryland, TR-544, May 1977.

[3] V. Wong, "Computational Structures for Extracting Edge Features from Digital Images for Real-Time Control Applications," Ph.D. thesis, Division of Engineering, California Institute of Technology, Pasadena, California, February 1979.

[4] L.S. Davis, "A survey of edge detection techniques," Computer Graphics and Image Processing, vol. 4, 1975.

[5] J. Tenenbaum, and H. Barrow, "Experiments in Interpretation Guided Segmentation," Artificial Intelligence Center, Stanford Research Institute, Palo Alto, California, Technical Note 123, March 1976.

[6] H. Freeman, "Computer processing of line-drawing images," Computing Surveys, vol. 6, no. 1, March 1974.

[7] H. Freeman, and L. Davis, "A corner-finding algorithm for chain-coded curves," IEEE Transactions on Computers, March 1977.

[8] J. Wilf, and R. Cunningham, Computing Arbitrary Moments from Chain-Coded Region Boundaries, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, June 1979.

[9] Y. Yakimovsky, "Boundary and Object Detection in Real World Images," Technical Memorandum 33-709, Jet Propulsion Laboratory, California Institute of Technology, 1974.

[10] I. Sobel, "Neighborhood coding of binary images for fast contour following and general binary array processing," Computer Graphics and Image Processing, Vol. 8, 1978.

[11] R. Wong, and E. Hall, "Scene matching with invariant moments," Computer Graphics and Image Processing, Vol. 8, 1978.

APPENDIX - FORMULAS FOR IMAGE STATISTICS

Let R be a region in an image with perimeter P = P(R), area A = A(R) and p,qth order moments Mpq = Mpq(R) = $\int_R X^p Y^q \, dxdy$. Let (X_i,Y_i) represent the ith point along R's boundary, where i = 0,...,n. Finally, let C_i be the chain code for the segment linking (X_{i-1},Y_{i-1}) with (X_i,Y_i), where i = 1,...,n and $C_i \in \{0,...,7\}$. Then, as in Fig. A-1:

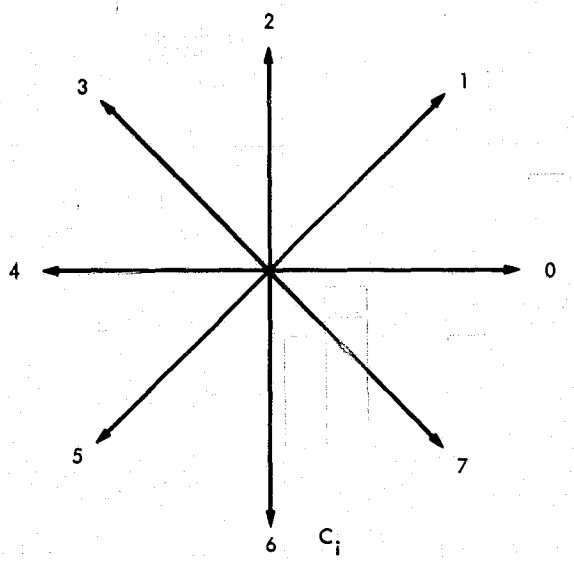


Figure A-1. Chain Code Directions and Labeling Conventions.

$$\Delta X_i = X_i - X_{i-1} = \begin{cases} 1, & \text{if } C_i \in \{0,1,7\} \\ 0, & \text{if } C_i \in \{2,6\} \\ -1, & \text{if } C_i \in \{3,4,5\} \end{cases}$$

$$\Delta Y_i = Y_i - Y_{i-1} = \begin{cases} 1, & \text{if } C_i \in \{5,6,7\} \\ 0, & \text{if } C_i \in \{0,4\} \\ -1, & \text{if } C_i \in \{1,2,3\} \end{cases}$$

$$P = \sum_{i=1}^{n} L_i \quad \text{where} \quad L_i = \begin{cases} 1, & \text{if } C_i \in \{0,2,4,6\} \\ \sqrt{2}, & \text{if } C_i \in \{1,3,5,7\} \end{cases}$$

$$A = M_{oo} = \sum_{i=i}^{n} \Delta Y_i X_i - 1/2 \Delta X_i \Delta Y_i$$

$$M_{10} = 1/2 \sum_{i=1}^{n} \Delta Y_i X_i^2 - \Delta X_i \Delta Y_i X_i + 1/3 \, \Delta X_i^2 \Delta Y_i$$

$$M_{01} = -1/2 \sum_{i=1}^{n} \Delta X_i Y_i^2 - \Delta X_i \Delta Y_i Y_i + 1/3 \, \Delta X_i \Delta Y_i^2$$

$$M_{02} = -1/3 \sum_{i=1}^{n} \Delta X_i Y_i^3 - 3/2 \, \Delta X_i \Delta Y_i Y_i^2 + \Delta X_i \Delta Y_i^2 Y_i - 1/4 \, \Delta X_i \Delta Y_i^3$$

$$M_{20} = 1/3 \sum_{i=1}^{n} \Delta Y_i X_i^3 - 3/2 \, \Delta X_i \Delta Y_i X_i^2 + \Delta X_i^2 \Delta Y_i X_i - 1/4 \, \Delta X_i^3 \Delta Y_i$$

$$M_{11} = 1/2 \sum_{i=1}^{n} \Delta Y_i X_i^2 Y_i - \Delta X_i \Delta Y_i X_i Y_i + 1/3 \, \Delta X_i^2 \Delta Y_i Y_i - 1/4 \, X_i^2 \Delta Y_i^2$$

$$- 1/2 \, \Delta Y_i^2 X_i^2 + 2/3 \, \Delta X_i \Delta Y_i^2 X_i$$

Using these moments, the region can be assigned a location and orientation. The location is defined by the region's center of mass, (X_{cm}, Y_{cm}). The orientation of the region is described, within a two-fold degeneracy, by the axis that minimizes second-order moments of inertia. Let θ be the angle between the X-axis and this axis of minimum inertia. Then,

X_{cm} = X - center of mass = M_{10}/M_{00}.

Y_{cm} = Y - center of mass = M_{01}/M_{00}.

θ = the angle of the axis that minimizes the moments of inertia.

$$\tan 2\theta = \frac{2 \, (M_{00}M_{11} - M_{10}M_{01})}{M_{00}M_{20} - M_{10}^2 - M_{00}M_{02} - M_{01}^2} .$$

NASA—JPL—Coml., L.A., Calif.