JPL PUBLICATION 84-66

JPL Robotics Laboratory Computer Vision Software Library

Robert Cunningham

November 1, 1984

NASA
National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No. 84-66	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Robotics Laboratory Computer Vision Software Library		5. Report Date November 1, 1984 6. Performing Organization Code
7. Author(s) Robert Cunningham		8. Performing Organization Report No.
9. Performing Organization Name and Address JET PROPULSION LABORATORY California Institute of Technology 4800 Oak Grove Drive Pasadena, California 91109		10. Work Unit No. BK-506-54-65-24-00 11. Contract or Grant No. NAS7-918 13. Type of Report and Period Covered JPL Publication External Report
12. Sponsoring Agency Name and Address NATIONAL AERONAUTICS AND SPACE ADMINISTRATION Washington, D.C. 20546		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

The past ten years of research on computer vision in the JPL Robotics Laboratory have matured into a powerful real-time system comprised of standardized commercial hardware, computers, and pipeline processing laboratory prototypes, supported by an extensive set of image processing algorithms. The software system has been constructed to be transportable via the choice of a popular high-level language (PASCAL), and a widely used computer (VAX-11/750). It comprises a whole realm of low-level and high-level processing software that has proven to be versatile for applications ranging from factory automation to space satellite tracking and grappling.

17. Key Words (Selected by Author(s))	18. Distribution Statement
Engineering (General) Mathematical and Computer Sciences (General) Computer Programming and Software	Unclassified-Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
	U	48	

JPL 0184 R 9/83

JPL PUBLICATION 84-66

JPL Robotics Laboratory Computer Vision Software Library

Robert Cunningham

November 1, 1984

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

The research described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

ABSTRACT

The past ten years of research on computer vision in the JPL Robotics Laboratory have matured into a powerful real-time system comprised of standardized commercial hardware, computers, and pipeline processing laboratory prototypes, supported by an extensive set of image processing algorithms. The software system has been constructed to be transportable via the choice of a popular high-level language (PASCAL), and a widely used computer (VAX-11/750). It comprises a whole realm of low-level and high-level processing software that has proven to be versatile for applications ranging from factory automation to space satellite tracking and grappling.

ACKNOWLEDGMENT

Many people have contributed to the work described in this document. Current members of the Robotics and Teleoperator Group include Bruce Bon, Thurston Brooks, Donald Gennery and Brian Wilcox. Former members of the Group who made substantial contributions include Raymond Eskenazi, Eric Saund, Yoram Yakimovsky and Melinda Yin. Acknowledgment is also due my technical supervisor, Edwin Kan, for his constant encouragement and direction.

CONTENTS

1. INTRODUCTION

This document describes our computer vision software library. The majority of the programs set forth represent the software base upon which we build programs to experiment with new algorithms and implement programs to demonstrate computer vision applications. Also described are some object tracking and acquisition programs representing current research (Section 7) and our implementations of camera calibration (Section 5). The software described here has evolved over varying amounts of time reflecting changes in hardware and algorithmic refinements based on our experience in using the programs. This document should be useful to the computer vision research and development community at large.

Most of the vision software was originally developed on a General Automation SPC-16/85 minicomputer in FORTRAN and assembly language. The majority of these programs have then been transferred in 1984 to a DEC VAX-11/750 computer running the VMS operating system. The few remaining programs will be transferred in the near future. FORTRAN programs transferred to the VAX have been converted to PASCAL and all new high-level software is written in PASCAL. Some low-level programs such as interrupt handlers and device I/O routines are written in assembly language (MACRO) as necessary.

The choice of a language for program development on the VAX was based on the desire to use a language with modern control statements (IF-THEN-ELSE, DO-WHILE loops, REPEAT-UNTIL loops, CASE statements), structured data types and strong variable typing. At the same time, we wanted our software to be transportable. Given the current lack of standardization of programming languages and operating systems, full transportability (i.e., running programs on another computer with

no modification) is practical only between two like computers running the same operating system. Beyond that, the best that can be hoped for is that programs can be made to run on a different computer with minimal changes. In that case, it is much easier to modify programs written in a structured language such as PASCAL (as opposed to FORTRAN for example) without introducing undesirable side effects that are difficult to correct. The logical choice was thus PASCAL which has the required language features and is supported by DEC to run under VAX/VMS.

While strictly speaking our software is fully transportable only to another VAX (11/750 or any other model), the fact that PASCAL is supported by most other modern operating systems means that transfer to a different computer should require minimal conversion effort.

As a final note on languages, it appears highly likely, if not certain, that ADA will be in wide use by the end of this decade, replacing PASCAL and other high-level languages, particularly on large computers such as the VAX. Since ADA is based largely on PASCAL, conversion to ADA would be relatively painless should we elect to do that in the future. It is interesting to note that many of the PASCAL language extensions provided by DEC resemble standard features of ADA. This is particularly true of the "environment/inherit" interface which links separately compiled PASCAL modules in a manner which is roughly equivalent to the "package" facility in ADA.

The following sections describe the vision system hardware and present the vision system software in an order which generally proceeds from low-level to high-level. Section 2 details the essential hardware components of the vision system. Section 3 describes low-level routines associated with image I/O

and hardware control. Section 4 sets forth the image feature extraction algorithms. Section 5 describes the camera calibration model and routines associated with calibration. Chapter 6 details coordinate transformation routines which use the camera calibration model to relate 2-D image points with 3-D points in a scene. Section 7 describes several object tracking algorithms and an automatic object acquisition program for tracker initialization.

2. VISION SYSTEM HARDWARE

The Robotics Laboratory hardware architecture is illustrated in Figure 2-1. The following sections describe various components.

2.1 VAX

The VAX-11/750 has 2M byte of memory, 180M bytes of disk storage (160M byte fixed disk and two 10M byte removable cartridge disks), 16 serial ports, two DR11-W 16-bit parallel ports, and a floating point accelerator. As resources are available we plan to add a tape drive, a 450M byte disk drive, and additional memory.

The vision hardware is installed on a QBUS connected to the VAX UNIBUS with a bus repeater. The bus repeater is transparent at the software level so the QBUS peripherals are programmed as if they were actually plugged into the UNIBUS.

2.2 Cameras

We are using three Hitachi KP-120 CCD cameras. The CCD array in these cameras is 244 lines by 320 pixels per line. The cameras are run in non-interlaced mode which provides a full image every 1/60 second. Two of the cameras are mounted as a stereo pair approximately six feet apart and oriented such that the common field of view is at a range of 6 feet. The third camera is mounted midway between the other two, approximately 3 feet higher. This arrangement will be used for 3-camera stereo experiments.

2.3 Digitizer

The digitizer is a DATACUBE QAF-120 which digitizes at 6.1

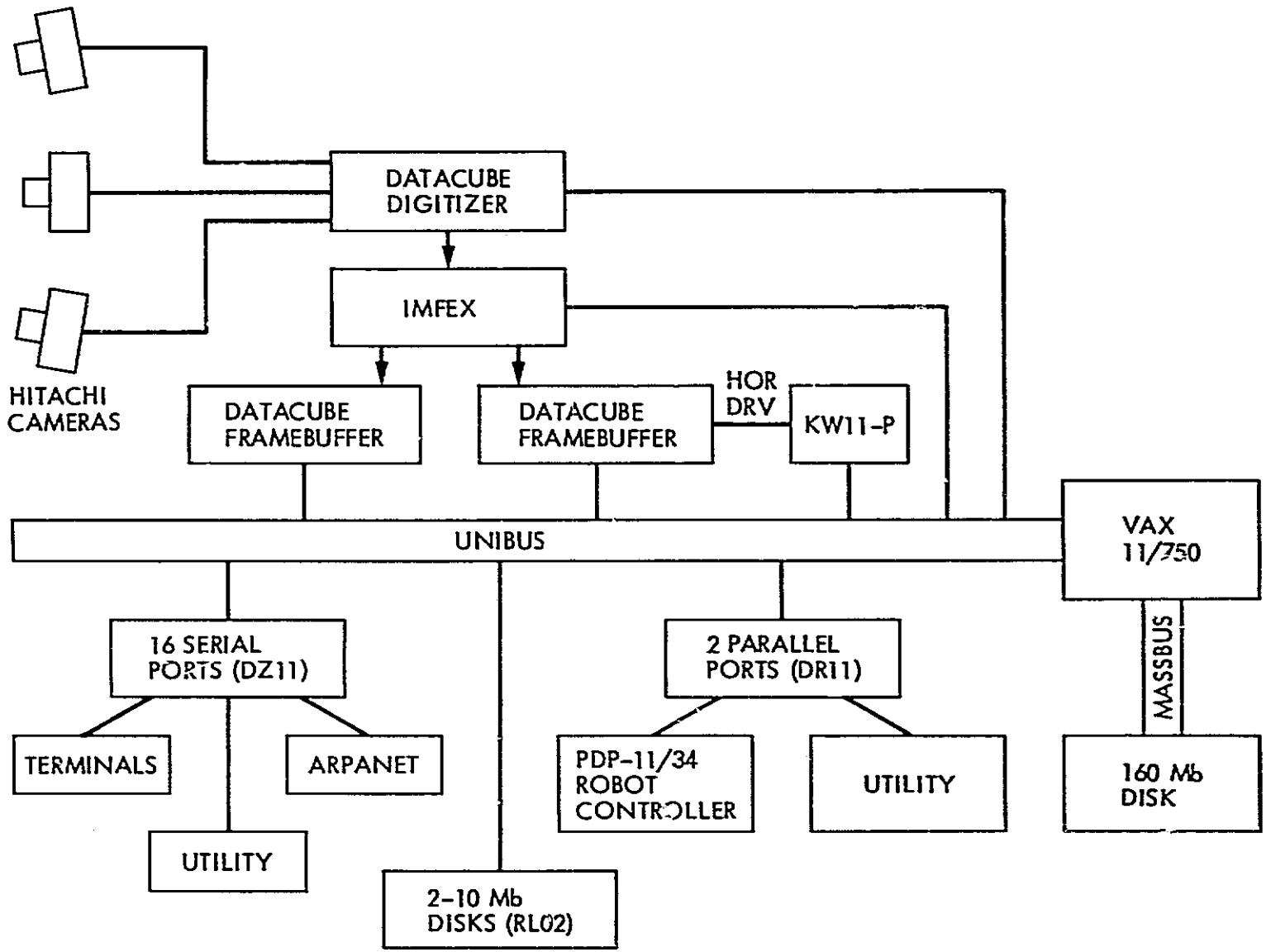


Figure 2-1. Vision System Hardware Architecture

MHz, the pixel rate of the Hitachi cameras. The digitizer can select one of four video inputs to be digitized.

2.4 IMFEX

The digitized video is input to IMFEX, a pipeline processor designed and built at JPL [1]. One of seven IMFEX outputs is selected to be stored in the frame buffers. These outputs include the raw digitized video, one of four stages of an edge detector, and two forms of binary images (thresholded video). The algorithms built into IMFEX are described in section 4.3.

2.5 Frame buffers

The frame buffers are DATACUBE QVG-120's, designed as companion products to the QAF-120 digitizer. In acquire mode, the frame buffers continuously store full images at the video rate; i.e., 1/60 second per image. In computer access mode, an image is frozen in the buffer. Programs have random access to single pixels for read/write operations by specifying the row and column address of the desired pixel. For line oriented operations, auto-increment may be selected after specifying only the address of the first pixel.

2.6 KW11-P Clock

The KW11-P is used to generate interrupts at 60 Hz so that programs can be synchronized to the video frame rate. The horizontal drive signal from one frame buffer is connected to the external clock input on the KW11-P. The KW11-P is programmed to interrupt every 262 horizontal drives (one frame time). Interrupt routines in the VAX perform functions such as counting frames, switching digitizer inputs, and switching frame buffers

between acquire mode and computer access mode.

3. IMAGE UTILITIES

The routines described in this section are low-level utilities which provide computer access to image data and some simple graphics capabilities for displaying the results of image processing.

The following standard arguments are used by the routines described in the following sections.

- I - pixel column address, $0 \leq I \leq 319$.
- J - pixel row address, $0 \leq J \leq 239$.
- G - pixel gray level, $0 \leq G \leq 255$.
  The notation G(I.J) stands for the gray level of pixel (I.J).
- FB - frame buffer number (0, 1, ...).
- CAM - camera number (1, 2, ...).
- LBUF - line buffer (320 byte array) which stores one line of image data.

Additional arguments are described as they occur in the text.

3.1 Initialization

Vision system peripherals on the QBUS are programmed by direct access to device registers to minimize software overhead. This requires mapping the corresponding UNIBUS I/O page into the program virtual address space. This represents the minimum initialization required to use the vision system. Additional initialization consists of programming peripherals with a default set of parameters. The initialization routines are:

FUNCTION MAPIO: INTEGER

performs the UNIBUS I/O page mapping.
Success or failure is indicated by an odd or even return value, respectively.

PROCEDURE INITVISION

calls MAPIO and then programs peripherals (digitizer, framebuffers, IMFEX) to a standard initialization state.

3.2 Frame Buffer I/O

These routines transfer image data between the VAX and the Datacube frame buffers.

FUNCTION GETPNT(FB,I,J):INTEGER

returns the gray level of pixel (I,J) in frame buffer FB.

PROCEDURE GETLINE(FB,J,LBUF)

loads image data from line J of frame buffer FB into the array LBUF.

FUNCTION GETNEXT(FB):INTEGER

returns the gray level of the "next" location in frame buffer FB; i.e., after GETPNT (FB,I,J), successive calls to GETNEXT return G(I+1,J). G(I+2,J),... Upon reaching the end of a line, the next pixel location is (0,J+1).

PROCEDURE PUTPNT(FB,I,J,G)

stores gray level G in location (I,J) of frame buffer

FB.

PROCEDURE PUTLINE(FB,J,LBUF)

copies the array LBUF to line J in frame buffer FB.

PROCEDURE PUTNEXT(FB,G);

used to write successive frame buffer locations (see remarks under GETNEXT).

3.3 Disk I/O

Images may be stored in disk files and retrieved later with the routines described below. Disk I/O is strictly sequential on a line by line basis. Random access to input files can be achieved by first copying the image to a frame buffer. Note that these routines will operate on arbitrary sized images.

The following arguments are used by these routines:

FCB - pointer to a file control block.
Several files may be opened simultaneously, each with its own FCB.

NAME - string argument specifying file name.

TYPE - string argument specifying 'new' or 'old' (existing) file.

NL - number of lines in the image.

NC - number of columns in the image.

FUNCTION IMAGEOPEN(FCB,NAME, TYPE, NL, NC): BOOLEAN

opens an image file for input ('old') or output ('new'). Returned value indicates success (true) or failure (false). For new files NL and NC are defined by the calling program. For old files, IMAGEOPEN assigns the values stored in the image file header to

NL and NC.

FUNCTION LINEIN(FCB,LBUF): BOOLEAN

reads the next line of an 'old' image into LBUF. The calling program must keep track of line numbers. Returns true if the operation is successful. Returns false if an I/O error occurs such as an attempt to read beyond the end of file, or if the file type is 'new'.

FUNCTION LINEOUT(FCB,LBUF): BOOLEAN

writes the next line of a 'new' image into the disk file. The calling program must keep track of line numbers. Returns true if operation is successful. Returns false if an I/O error occurs or if the file type is 'old'.

IMAGECLOSE(FCB)

closes an image file. If the file type is 'new', this procedure writes the file header recording the actual number of lines written in the file.

3.4 Camera Selection

SELCAM(CAM NUM,FB)

selects camera CAMNUM (currently 1, 2, or 3) as input to digitizer, and puts frame buffer FB in acquire mode to store digitized video. CAM NUM=0 freezes frame buffer FB after allowing at least one full frame to be stored.

3.5 IMFEX Control

The routines described below are called to program IMFEX.

IMFOUTPUT (DISPCODE)

selects raw image data or one of the processed outputs to be passed to the frame buffer. For convenience, the display codes are defined as symbolic constants:

0 =	IMF_RAW	raw image.
1 =	IMF_GRAD	gradient.
2 =	IMF_THIN	thinning.
3 =	IMF_THRESH	thresholding.
4 =	IMF_LUT	lookup table.
5 =	IMF_BLOB	thresholded video.
6 =	IMF_BLOBLUT	LUT applied to thresholded video.

IMFTHRESH(T)

sets threshold in thresholding unit to T, $0 \leq T \leq 255$.

LOADLUT(TABLE)

loads the IMFEX lookup table from the array TABLE. TABLE[i] = 0 or 1, $0 \leq i \leq 511$.

READLUT(FNAME, TABLE)

reads a lookup table from an ASCII disk file FNAME into the array TABLE.

3.6 Cursor

A cursor is generated in hardware and mixed with video displays as a cross hair. The cursor may be displayed at an arbitrary location I,J to indicate the performance of a program such as an object tracker. The cursor may also be positioned manually (keyboard, joystick) and its position returned to a program allowing the operator to designate points or regions of interest. There is also a software blinking cursor which is displayed as a 5x5 "+", alternately written into the frame buffer as black (0) and white (255).

PUTCUR(I,J)

display the hardware cursor at image location (I,J).

CURSOR(I J[,FB])

allows operator to position the cursor (software or hardware) at a desired location. Typing "escape" terminates the procedure with the location returned in I,J. The optional FB argument specifies which frame buffer to use for the software cursor. The default value is 0.

3.7 Graphics

Graphics routines are used to display the results of image analysis routines (for demonstration and debugging purposes). Additional displays are generated as necessary using PUTPNT/PUTLINE.

DRAWLINE(FB,I1,J1,I2,J1,G)

draws a "straight" line between image points (I1,J1) and (I2,J2) in frame buffer FB. The line is displayed at intensity G.

PUTCHAR(FB,C,I,J[,G])

displays a single character C at I,J in frame buffer FB. Each character is generated in a 5x7 dot matrix. I.J refers to the center of the matrix.

PUTSTRING(FB,S,I,J[,G])

displays a string of characters in frame buffer FB starting at image location I,J.

3.8 Image Utility Program

The program PICTURE is a stand-alone utility program which allows one to interactively perform a variety of image-related operations. Commands are provided to make individual calls to many of the routines described in this chapter (e.g., SELCAM, CURSOR, DRAWLINE, IMFOUTPUT, etc.) allowing convenient and flexible control of the vision system for test purposes. The program is also useful for saving images (real or artificial) to be used as control data sets during program development and debugging.

The program provides access to image data in one of the frame buffers, a disk file, or an internal "picture" array. The picture array buffers image data in floating point format; i.e., intensity levels are represented as single precision floating point numbers (32 bits). Operations such as smoothing, adding random noise, and generating artificial images are performed on the picture array using floating point arithmetic.

Each command is a single-character, including a help command ("?") which displays a menu of the available commands on the terminal. Some of the functions provided enable the operator to:

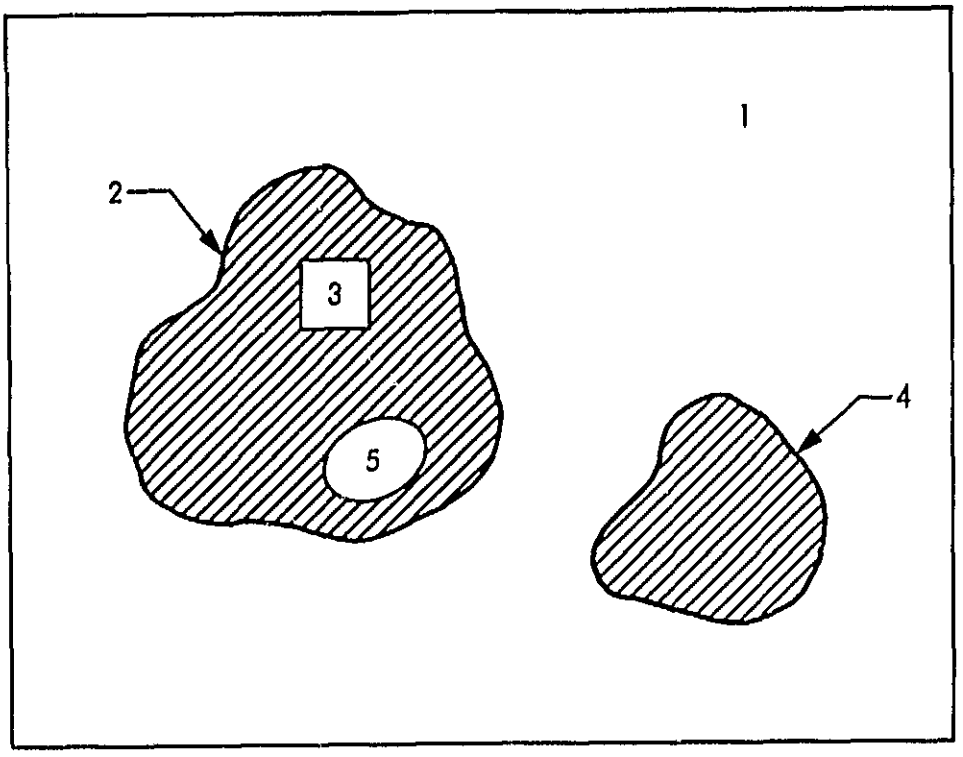
1. Move data between any two storage areas (frame buffer, disk file, picture array).
2. Select and freeze (unfreeze) a frame buffer.
3. Select digitizer video input.
4. Program IMFEX.
5. Add random noise to an image.
6. Generate an artificial image.

4. FEATURE EXTRACTION

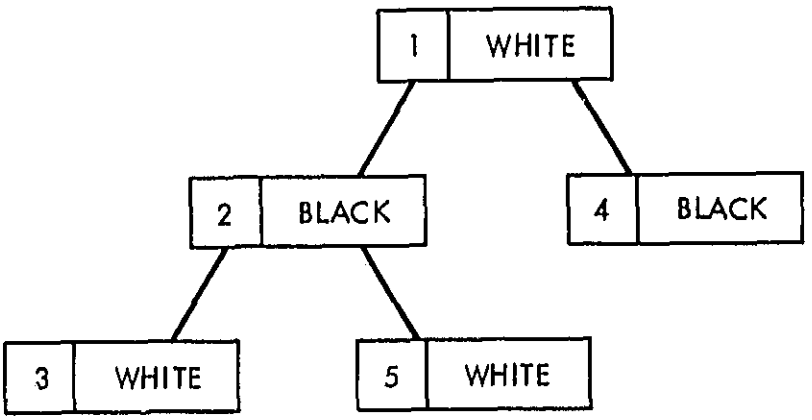
This section describes routines which extract image features based solely on the 2-D image array data. Routines of this type generally fall into one of two categories: clustering algorithms such as region growers and convolution algorithms such as edge detectors. Clustering algorithms extract regions from images based on a property such as intensity which is common to neighboring pixels. Convolution algorithms transform the image array (to enhance contrast edges for example) by performing some operation defined for a small window (3x3, 5x5, etc.) centered at each pixel in the image. Also included in this chapter is a stereo correlation algorithm which is essentially a convolution operation (gray level correlation) although the operation of the algorithm is constrained by the 3-D camera model.

4.1 Blob Extraction

The routine BLOBEXT scans a binary image (obtained by thresholding) to produce a "blob tree" description of an image. Each node in the blob tree represents a connected region of uniform color (black or white). The hierarchy of the tree represents containment; i.e., nodes at level K represent blobs which are completely surrounded by the blob represented in the parent node at level K-1. Figure 4-1 shows a simple binary image and the corrresponding blob tree. The approach used here is similar to that of the SRI Vision Module [2].



BINARY IMAGE



BLOB TREE

Figure 4-1. Blob Tree Representation of Binary Image

Several statistical descriptors are obtained for each blob and stored in the corresponding node of the blob tree:

Area
Perimeter
1st moments ($\sum I, \sum J$)
2nd moments ($\sum I^2$, $\sum IJ$, $\sum J^2$)
Minimum enclosing rectangle (Imin, Imax, Jmin, Jmax)
Number of holes (blobs bordering and surrounded by this blob)
Color (black/white)

BLOBEXT also produces a run-length encoding of the image. This consists of a table of run-length record lists, one list per line of the image. Individual run-length records represent chains of adjacent pixels which are all white or all black. Specifically, a run-length record contains 3 pieces of information:

1. left-most column address I.
2. length of the run.
3. color.

4.2 Region Grower

The region grower routine REGGROW is based on the "blooming" region grower described in [3]. Each call to the region grower extracts a single region represented by a record similar to a blob tree node defined above.

The region grower is called with the coordinates of a "seed" point which by definition belongs to the region. Each of the 4 - or 8 - connected nearest neighbors are placed in a candidate queue. Candidates are removed from the queue one at a time. If

the candidate passes an acceptance test (see below), the point is added to the region and its unprocessed neighbors are placed in the queue. Processing continues in this manner until the queue is empty. The region grower is constrained to a user-selectable rectangular window defined by the initialization routine INITGROW which must be called before the first call to REGGROW. REGGROW may be called multiple times to extract several non-overlapping region records in a given window.

An argument in the REGGROW call selects one of the four following acceptance tests:

1. G(I,J) < Threshold

2 G(I,J) > Threshold

3 Accept point (I,J) as long as $G_{max} - G_{min} <$ Threshold

4. Accept point (I,J) if $\frac{(G(i,j) - \mu(i,j))}{\sigma^2 \ (i,j)} < T$

where (i,j) and (i,j) represent the local estimates of the average and standard deviation of gray levels inthe region.

4.3 Edge Detection

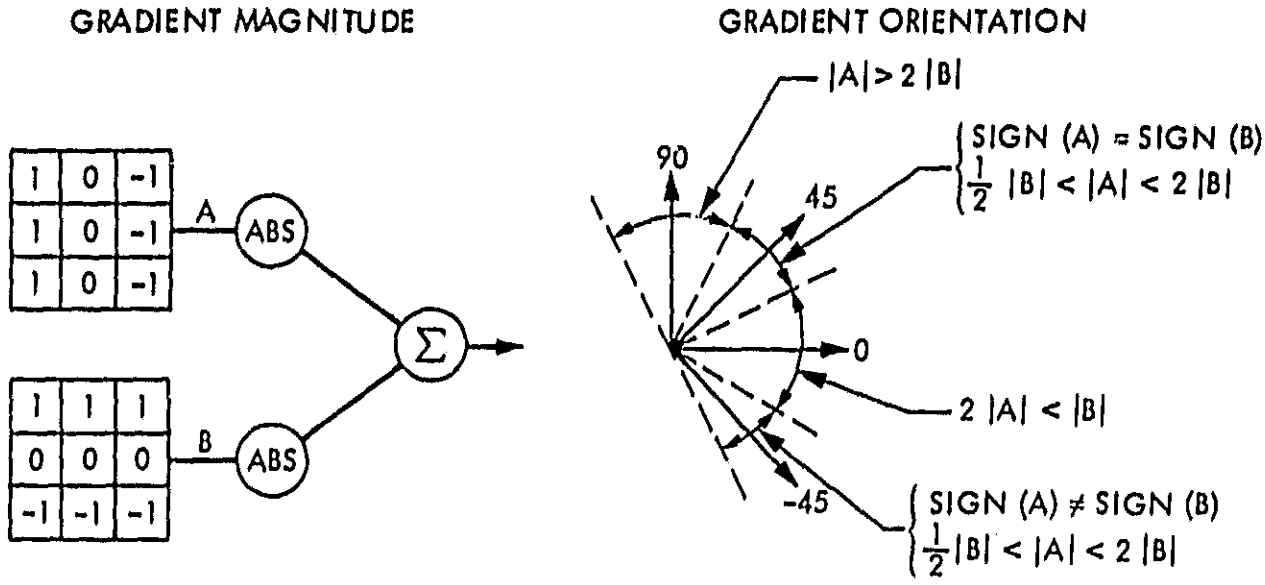
This section describes the processing performed by IMFEX. While these algorithms could be done in software, they are currently implemented in hardware. (The hardware algorithm runs in real time compatible with the camera frame rate of 60 hertz, while the software implementation is much slower.) A description is included here for completeness.

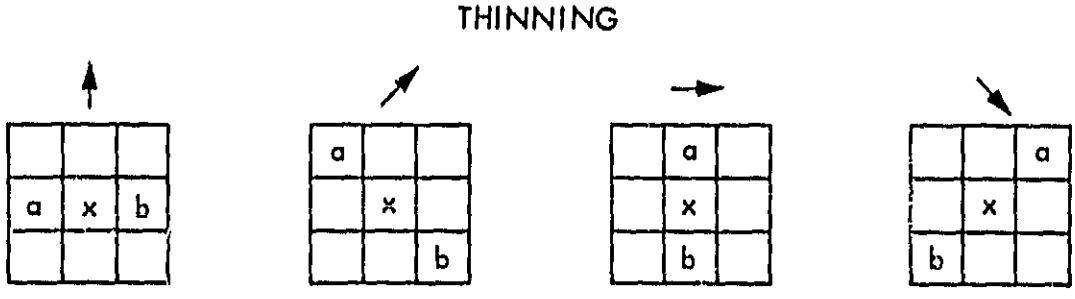
GRADIENT The gradient operator calculates the magnitude of the average gradient at each pixel and a coarse estimate of edge orientation. Two three by three convolutions are performed in parallel and summed. The absolute value of this sum represents the average gradient. The orientation is estimated by the equivalent of an arctangent operation (see Fig.4-2).

THINNING The thinning algorithm uses non-maximum suppression to eliminate edges by comparing the average gradient of each pixel to its two nearest neighbors orthogonal to the gradient orientation. The output of this step at each pixel is either the input gradient or zero.

THRESHOLDING This step converts the array of gradient values to a binary edge map by comparing to a threshold. The threshold is set in software using the routine IMFTHRESH. In an alternate mode of operation, the raw video can be thresholded to produce a binary image.

TABLE-LOOKUP The binary edge map (or image) can be further processed by a lookup-table algorithm. The nine cells of a 3x3 window are ordered so that each combination of edge/no edge (1/0) data corresponds to a unique 9-bit binary number (0-511). This number is used to address a 512 x 1 memory. The value stored in the addressed location replaces the center pixel of the corresponding 3x3 window. The lookup table can be programmed to remove isolated edges, fill single pixel gaps, perform further thinning, and detect patterns such as vertices.





IF MAX { x, a, b } = x THEN OUTPUT x ELSE OUTPUT 0.
(SELECT a, b ACCORDING TO ORIENTATION OF x)

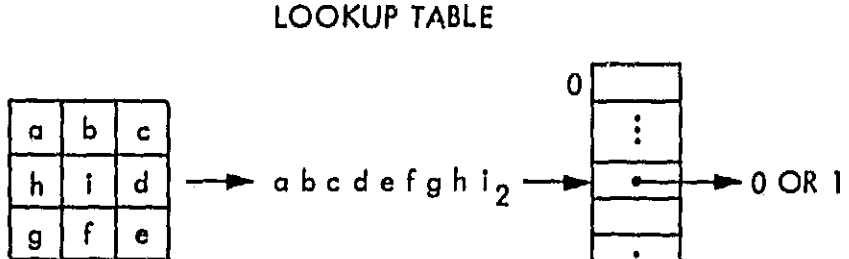


Figure 4-2. IMFEX Processing: Gradient Magnitude and Orientation, Thinning, Lookup Table

4.4 Stereo Correlation

The stereo correlation routine CORREL is used to obtain 3-D measurements of selected points in a scene [4]. Assume that a point on the surface of an object is visible in both cameras at image points (I_1,J_1) and (I_2,J_2) respectively. Given one of the image points, say (I_1,J_1), CORREL searches for the conjugate point (I_2,J_2) in the other image using correlation of gray levels as the matching function.

The search for a match is restricted to points along a line segment as shown in Figure 4-3. The search segment is defined by placing upper and lower bounds on the a priori estimated range to the object and projecting the corresponding segment of the ray $R(I_1,J_1)$ into the second image. Using a programmable sampling mask, the second image is sampled at each point along the search segment. Each sample is correlated with a spatially equivalent sample obtained from the first image using the standard statistical correlation coefficient formula. The correct match is identified as the point which maximizes the correlation coefficient.

Before beginning the search, an autocorrelation test is performed in the neighborhood of (I_1,J_1). The purpose of this test is to determine if there is a well defined peak which will produce an unambiguous local maximum correlation value in the second image. If the autocorrelation test fails, the mask is increased in size and the test repeated. This is done at most N times (N = 3 typically) and if the test fails in all cases, CORREL returns a failure code without attempting to find the matching point.

Assuming the CORREL is successful, the 3-D coordinates of the point are obtained by calling INTSCT (section 6).

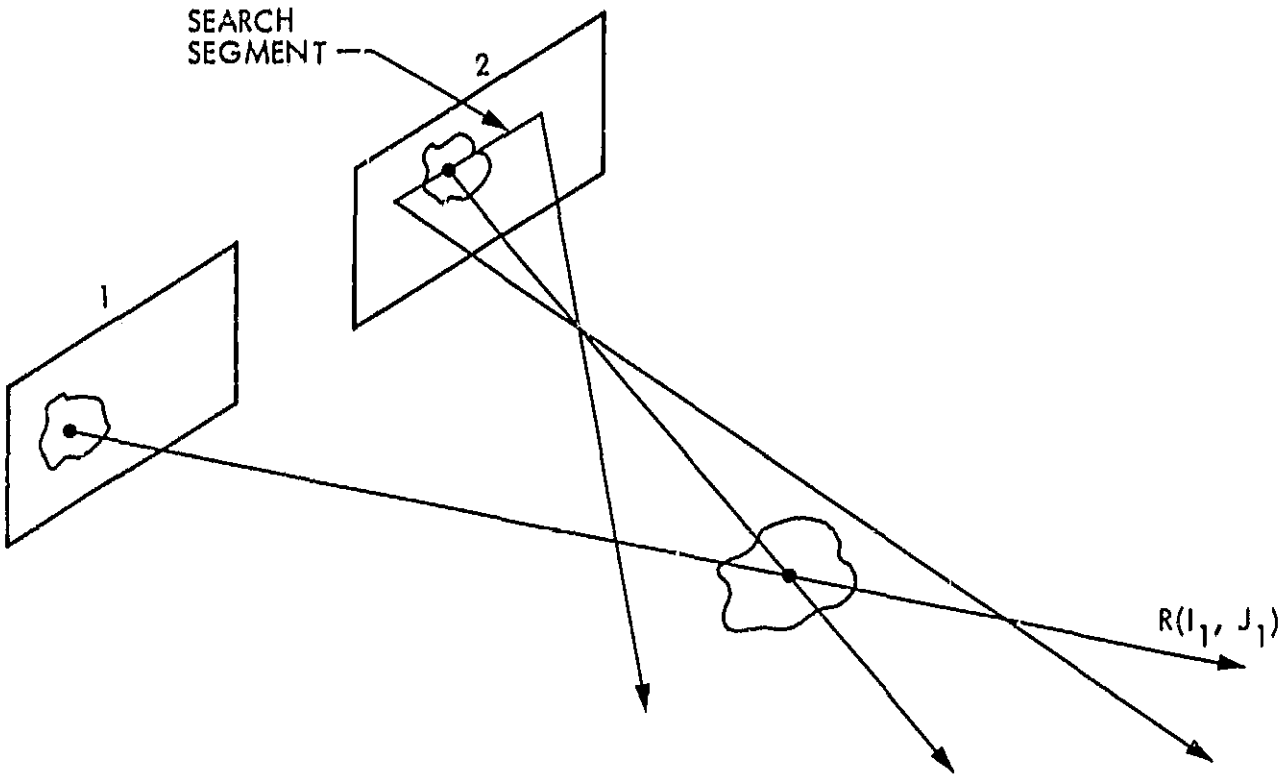


Figure 4-3. Search Segment for Stereo Matching

5. CAMERA CALIBRATION

This section describes the camera model we use to make 3-D measurements, and the procedure for calibrating the cameras.

The cameras are calibrated empirically by viewing several known 3-D points and recording the coordinates of the corresponding image points. These data are then used to solve for the camera model parameters using least-squares techniques.

One advantage of this approach is that it is not necessary to have prior knowledge of camera parameters such as the 3-D location of the lens center, orientation of the principal axis, focal length, and the location of the image plane in the reference 3-D coordinate frame. This is a very important consideration when using non-photogrammatic cameras since direct measurement of these parameters can be extremely difficult to obtain. Another advantage relates to stereo matching. Using the camera model, the line in one image plane containing all possible match points for a given point in the other image can be determined without requiring that the image planes be coplanar with precise line to line registration.

The accuracy of the camera model is tested by comparing the observed image coordinates of a known 3-D point to the values computed with the calibration parameters. Typically the average error is 0.1 pixel with a worst case error of 0.4 to 0.5 pixel. These error estimates reflect sub-pixel accuracy measurements obtained by calculating the centroid of a region covering approximately 100 pixels. However, they indicate at most a 1/2-pixel error for discrete measurements based on a single pixel.

5.1 Camera Model

The camera model is based on a standard perspective transformation which projects points in 3-D space through a single point, the lens center, onto the image plane [4]. The form that we use is defined by two equations which give the horizontal (I) and vertical (J) components of the projection of a 3-D point X:

$$I = \frac{(X-C,H)}{(X-C,A)}$$

$$J = \frac{(X-C,V)}{(X-C,A)}$$

where C is the position of the lens center in a reference 3-D coordinate system, A is the principal axis of the lens, H is the "horizontal" vector and V is the "vertical" vector. Note that (X-C,H) represents the dot product of the vectors X-C and H.

The geometrical interpretation of these vectors is illustrated in Fig. 5-1. C is expressed in mm, A is a unit vector, and H and V are scaled so that I and J are in "pixel" units. Figure 5-1 is simplified by showing the image coordinate origin in the center (i.e., the projection of A) and showing A, H and V as mutually orthogonal. The image origin is actually located at the upper left corner to agree with frame buffer addressing with the result that H and V are skewed to compensate for the translation.

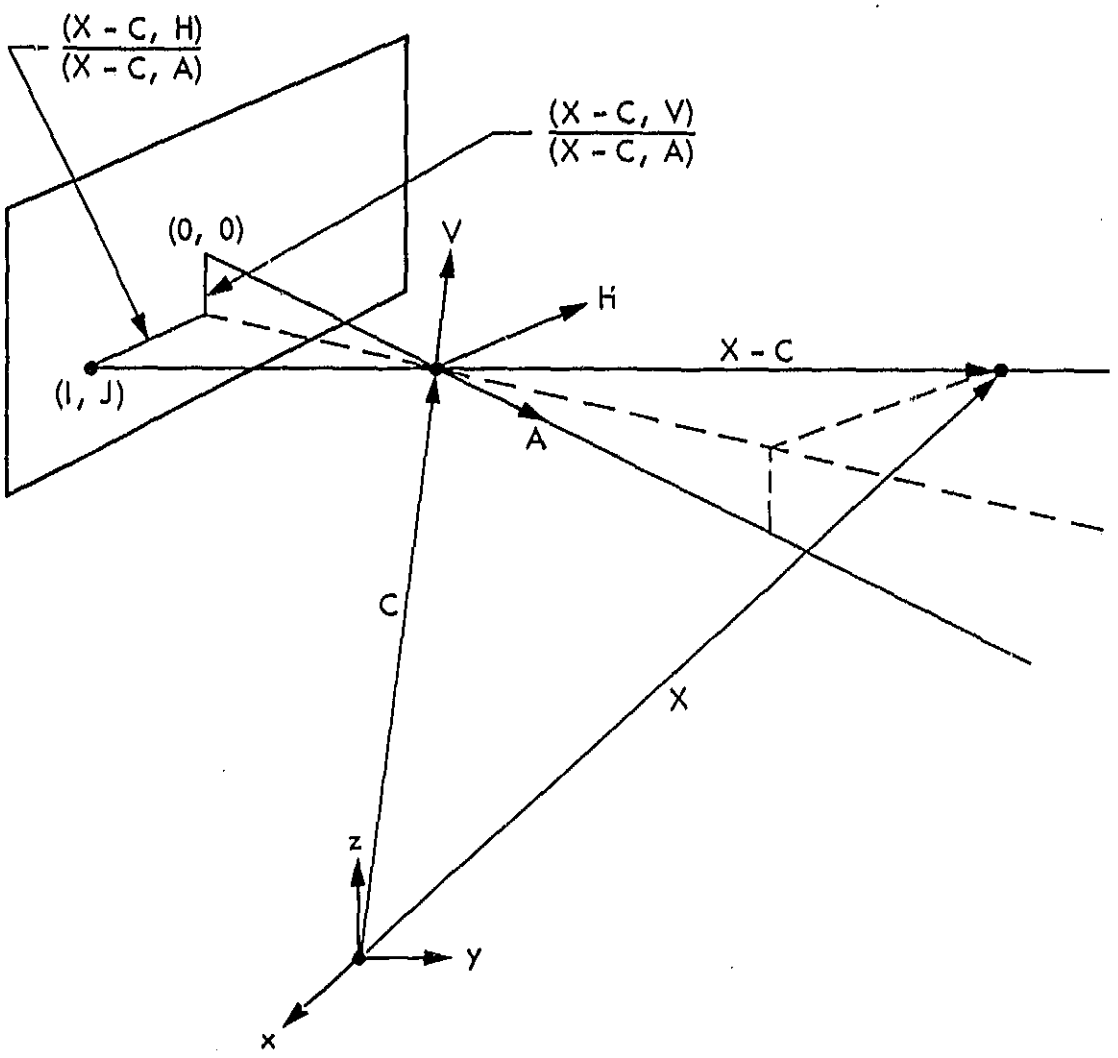


Figure 5-1. Camera Calibration Model

5.2 Calibration Solution

The camera model is obtained by the program SOLVECAL which computes values for the vectors C, A, H, and V. The input to this program is a list of N measurements $P_k = (I_k, J_k, x_{k1}, x_{k2}, x_{k3})$, each representing the observed image (I_k, J_k) of a known 3-D point X_k. The minimum number of measurements required is 6. However, we typically obtain 100 to 200 measurements and solve for the calibration vectors using least-squares techniques. Also the 3-D points X_k may not all be coplanar. Each X_k is used to write two equations based on the defining equations given in the previous section:

$$Ix_{k1}a_{k1}+Ix_{k2}a_{k2}+Ix_{k3}a_{k3}-I(C,A)-x_{k1}h_{k1}-x_{k2}h_{k2}-x_{k3}h_{k3}+(C,H) = 0$$
$$Jx_{k1}a_{k1}+Jx_{k2}a_{12}+Jx_{k3}A_{k3}-J(C,A)-x_{k1}v_{k1}-x_{k2}v_{k2}-x_{k3}v_{k3}+(C,V) = 0$$

These equations are obtained by rewriting the equations on I and J so that all terms appear on the left hand side, and then expanding the dot products. In this manner we obtain a homogeneous system of 2*N equations in the 12 unknowns

$$a_1, a_2, a_3, h_1, h_2, h_3, v_1, v_2, v_3, (C,A), (C,H), (C,V).$$

In order to solve this system of equations, it is necessary to assign an arbitrary value to one of the unknowns so that we get a nonhomogeneous system of 2*N equations in 11 unknowns. For this purpose, the operator selects one component of A to be assigned the value 1. The selected component should be the one which appears to be closest to 1 based on the orientation of the camera with respect to the reference 3-D coordinate frame.

After the initial system of equations is solved, the program solves for C using the following 3x3 system of equations:

$$c_1a_1 + c_2a_2 + c_3a_3 = (C,A)$$
$$c_1h_1 + c_2h_2 + c_3h_3 = (C,H)$$
$$c_1v_1 + c_2v_2 + c_3v_3 = (C,V)$$

Finally A, H and V are scaled by $1/|A|$ with the result that the final value of A is a unit vector.

The solution is verified by comparing the observed I_k's and J_k's with the values predicted by the camera model:

$$I_k = I_k - \frac{(X_k - C,\ H)}{(X_k - C,\ A)}$$

$$J_k = J_k - \frac{(X_k - C,\ V)}{(X_k - C,\ A)}$$

If either $|I_k|$ or $|J_k|$ exceeds a threshold, P_k is removed from the data set, and a new solution is obtained. This cycle is repeated until all errors are less than the operator specified threshold, typically around 1.

5.3 Calibration Data Acquisition (CALDATACQ)

The camera calibration data acquisition program uses the region grower (Section 4.2) to locate a target attached to the end of a PUMA-600 robot. The target may be any easily detected figure whose centroid position is known relative to the position of the PUMA. We use a black disk drawn on white paper attached to the center of the flange at the end of the robot. The centroid of the disk coincides with the position of the robot calculated by VAL, the PUMA controller.

CALDATACQ moves the PUMA through a sequence of 128 locations so that the target is positioned at each point of two 8x8 grids contained in distinct planes. At each position, the target location x, y, z and the corresponding image centroid obtained by the region grower are recorded in a disk file which is input to SOLVECAL later. The target grids are defined interactively as described below. Data acquisition then proceeds automatically.

GRID DEFINITION

The operator is prompted to move the PUMA to four locations so that the target appears near the upper-left, upper-right lower-right and lower-left corners of the image. At each location, the operator positions a cursor on the target and the program records the cursor position (I,J) and the target position (x,y,z). This defines two quadrilaterals, one in the image plane and one in 3-D space.

The program then effectively defines two corresponding 8x8 grids, one in 3-D space and one in the image plane, by calculating 8 equally spaced points including end points on each side of the quadrilaterals defined by the two sets of four corner points. The grid points are determined by the intersections of lines connecting corresponding points on opposite sides of the quadrilaterals.

DATA ACQUISITION

The data acquisition loop moves the robot to position the target at each 3-D grid point. The program then attempts to locate the target by sampling the image at the corresponding image grid point. If the sampled point is black, it is assumed to be in the target disk. If the point is not black, a spiral

search is initiated which terminates successfully when a black point is located or unsuccessfully if an error bound of 10 pixels is exceeded, in which case the operator is prompted to position a cursor on the target. The region grower is then called starting at the point determined (by one of the above means) to be on the target to locate the entire disk and calculate its centroid.

Each time the region grower finishes the operator is prompted to accept the result or try again with a higher or lower acceptance threshold in the region grower.

The grid definition/data acquisition sequence is then repeated for the second 8x8 grid. The entire process may then be repeated for additional cameras as desired.

5.4 CALIBRATION DATA FILE MANAGEMENT

User programs requiring camera calibration data acquire it from files created and updated by the program NEWCAMCAL. The standard calibration file contains data for each of the cameras being used in the laboratory. A data record for each camera contains the calibration vectors C, A, H and V, a name (i.e. "LEFT", "RIGHT"), and the image size.

NEWCAMCAL is normally run after SOLVECAL to update the standard calibration file. It can also be used to create alternate files. Some uses of the latter capability are 1) generating artificial calibration data for test purposes, and 2) creating calibration files corresponding to images taken by a camera other than the ones in our lab.

User programs access calibration data by calling the procedure SETCAMCAL which loads the standard calibration file or SETCAMCALUSER(FILENAME) which loads an alternate calibration file as specified by the FILENAME argument.

6. COORDINATE TRANSFORMS

The routines described in this section perform a variety of coordinate transform functions involving the camera calibration model. SETCAMCAL or SETCAMCALUSER must be called to bring calibration data into the program before calling any of these routines.

The two basic functions are XYZToImage which computes the image (I,J) of an arbitrary 3-D point X = (x,y,z), and ImageToRay which calculates the ray R(I,J) containing all 3-D points whose image is (I,J). Additional functions are defined by imposing various 3-D constraints.

These routines use the standard arguments listed below. Reals are single precision.

I = image column coordinate (real)
J = image row coordinate (real)
CAM = camera number (integer 1,2,...)
R, X, P, N = 1-dimensional array of three reals (x,y,z)

XYZToImage (CAM, I, J, X)

calculates the projection (I,J) of the 3-D point X into camera CAM.

ImageToRAY (CAM, I, J, R)

calculates a unit vector R representing the projection of image point (I,J) in camera CAM into 3-D space. The ray defined by R denoted by R(I,J) contains all 3-D points whose image is (I,J).

ImageToXYZ(CAM, I, J, CONSTRAINT, X[, P, N])

calculates the position X of a point imaged on (I,J) in

camera CAM given a CONSTRAINT (integer 1-4) which specifies a plane containing the point X. The specified plane may be parallel to one of the three coordinate planes (xy,xz,yz) or it may be an arbitrary plane defined by a point P in the plane and the normal to the plane N (unit vector). Specifically a CONSTRAINT of k=1, 2 or 3 specifies the plane X[k] = CONSTANT with the actual value stored in X[k] by the calling program (i.e., if k=3, then X is in the plane Z=X[k] parallel to the xy-plane). One use of this routine is to measure the position of an object resting on a level surface at a known height (z - coordinate).

If CONSTRAINT = 4, then P and N are defined by the calling program. The defaults are P = (0,0,0) and N = (0,0,1) which is equivalent to CONSTRAINT = 3, X[3] = 0. P and N are ignored if CONSTRAINT = 1,2 or 3.

INTSCT(CAM1,CAM2,I1,J1,I2,J2,X)

is used to make 3-D measurements based on stereo triangulation. (I1,J1) and (I2,J2) are the images in CAM1 and CAM2 of a common 3-D point. The coordinates of this point are returned in X. The computed value of X corresponds to the midpoint of the segment which minimizes the distance between the two rays R(I1,J1) and R(I1,J1).

ImageToImage(CAM1,CAM2,I1,J1,I2,J2,MODE,D)

Given an image point (I1,J1) in CAM1, returns the coordinates (I2,J2) in CAM2 of the projection of a point P on the ray R(I1,J1). The point P may be specified in one of two ways according to the arguments MODE(integer) and D(real) as follows:

MODE=0 - P is at distance D from CAM1 along R(I1,J1). Note that if P projects outside of the CAM2 image, then I2 will be coerced to 0 or 319 as appropriate with (I2,J2) representing the projection of an alternate point P' on $R(I_1,J_1)$.

MODE=1 - P is defined to be the point on $R(I_1,J_1)$ whose image is (D,J2).

This routine is used by the stereo correlation routine CORREL to define the search segment for stereo matching (see Section 4.4).

7. TRACKING AND ACQUISITION

This section describes three object tracking programs and an object acquisition program. The first two programs (correlation tracker & label tracker) are based on fairly simple algorithms which execute in real-time (60 Hz). The third program tracks known 3-D objects using a robust model-based approach which computes six components of position and velocity (linear and angular) and tolerates noise and/or partial occlusion. The acquisition program is used to initialize the latter tracking program. The front end of the acquisition program is a real-time vertex tracker. Using rigid body constraints, the vertex position and velocity estimates lead to a model-matching procedure which establishes the position and velocity of the object to initialize the tracker.

Object tracking is an essential component of visual feedback in dynamically changing environments. Some applications are grasping moving objects or performing assembly operations on moving objects with a robot arm; real-time obstacle avoidance and landmark navigation by an autonomous vehicle; and spacecraft rendezvous and docking or berthing operations. The basic requirement in such applications is fast processing. For standard TV cameras, the maximum rate is 30 Hz (standard RS-170) or 60 Hz (non-interlaced).

Object tracking algorithms based entirely on software must be kept simple to execute at video rates on a general purpose computer. The correlation tracker and the label tracker fall into this category. More robust algorithms such as the 3-D acquisition and tracking algorithms will require much more computational power to execute at video rates. This includes powerful image processing hardware and parallel array processors for mathematical computations. IMFEX is an example of the kind

of image processing hardware necessary for real-time vision programs.

7.1 Correlation Tracker

The correlation tracker is a library module which can be linked to any application program. The tracker is initiated by the routine

```
CTRACK(I,J,CAM,FB1,FB2,FCTR)
    I,J - image coordinates (row,column) of object,
          initialized by caller, updated by tracker.
    CAM - camera number (1,2,...)
FB1,FB2 - frame buffers to be used (0,1,...)
   FCTR - frame counter incremented by tracker at 60 Hz.
```

Tracking is accomplished by correlating successive images with the initial gray level sample centered at the point (I,J) in the call to CTRACK. The search in each image is centered at the next predicted location which is the last observed location plus I,J offsets based on velocity estimates. Velocity is estimated independently in the I and J directions as the slope of a least squares fit line using the last five observations.

The correlation sampling mask consists of the 25 points on the horizontal, vertical and diagonal bisectors of a 7x7 square. The search consists of 25 samples centered at the points of a 5x5 square which in turn is centered at the predicted I,J. The updated I, J represents the point where the correlation is maximized.

A new iteration of the tracker is initiated every 1/60 second using the KW11-P interrupts. Images are alternately frozen in FB1 and FB2 to access images at 60Hz. Two frame

buffers are required for this scheme since computer access is blocked while image data is being stored in the frame buffer.

The actual tracking procedure is an Asynchronous System Trap routine (AST) invoked by the KW11-P interrupt routine. The variable FCTR is also incremented by the KW11-P interrupt routine so that the calling program can synchronize processing with the tracker. This routine was used to demonstrate real-time vision-based navigation of the JPL Mars Rover test vehicle [5].

7.2 Label Tracker

The label tracking program was developed for the general case of providing 3-D vision feedback to a robot arm manipulating a moving object [6]. We are using it to conduct laboratory experiments in tracking and capturing a moving satellite with the PUMA robot [7]. 3-D measurements obtained by the tracker at 30 Hz are used to estimate the trajectory of the satellite. The PUMA is commanded to follow a similar trajectory which keeps the end effector poised near a grapple fixture on the satellite. This continues indefinitely until the operator issues the grapple command. Upon receiving this command the PUMA begins to approach the satellite until the end effector attaches to the grapple fixture. The satellite is then slowed gradually until it comes to rest. PUMA control software resides in a PDP-11/34. A DR11-W parallel interface on the VAX is used for communication to the PDP-11/34.

The tracking algorithm requires three labels arranged as the vertices of a right trangle at known locations on the object (Figure 7-1). In the satellite capture experiment the labels are white disks on a black background. The centroids of the three labels define the x and y axes of an object centered coordinate frame, with the z-axis obtained by their cross

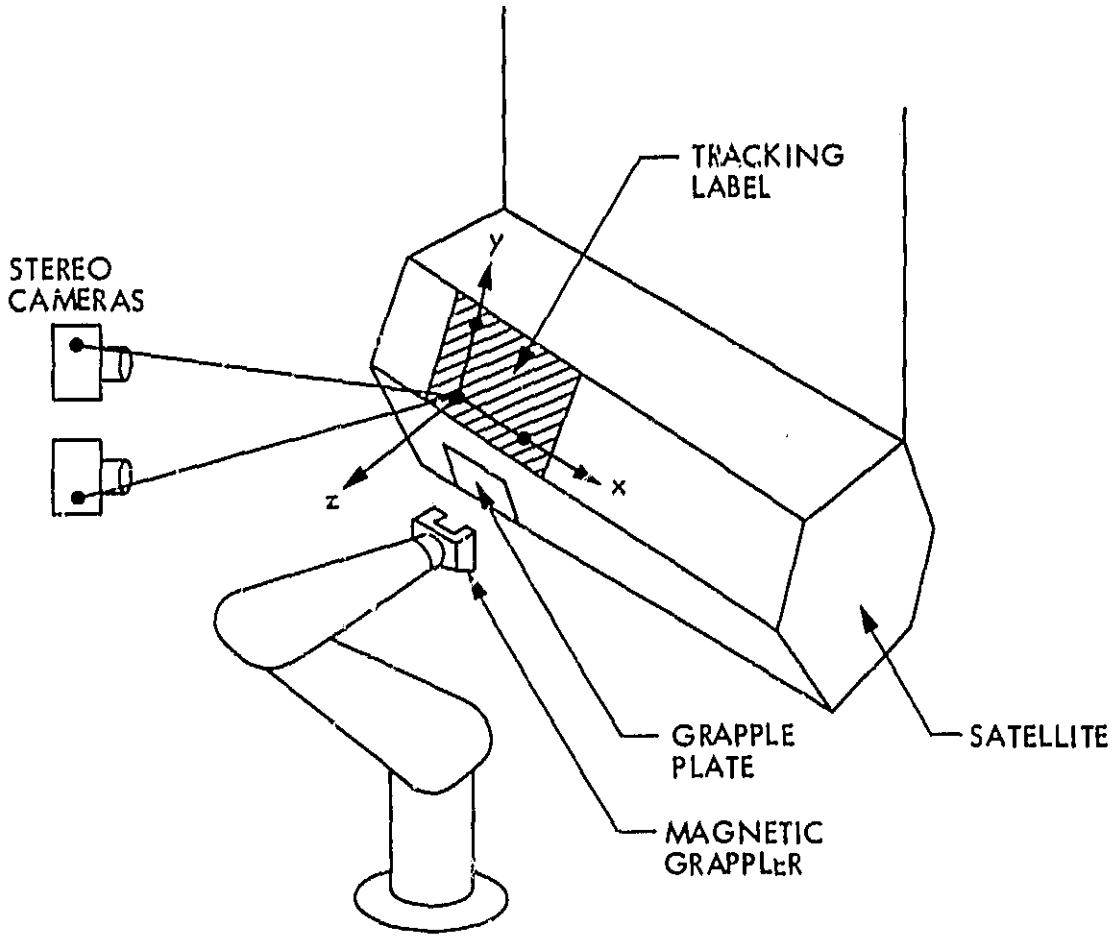


Figure 7-1. Laboratory Setup for Satellite Tracking Experiment

product. The position and orientation of the object are represented by a homogeneous coordinate transformation matrix ${}^{obj}T_{base}$ (read "object" with respect to "base"). This representation is used to describe the location of the object since it also is the representation used to specify the desired location of the end effector ${}^{hand}T_{base}$ in our robot control software. In general, the desired end effector position will be displaced from the origin of the object (one obvious reason is to avoid occlusion of the labels by the robot). This displacement is represented by a third coordinate transformation ${}^{hand}T_{obj}$. The desired end effector position is calculated as

$$ {}^{hand}T_{base} = {}^{hand}T_{obj} \cdot {}^{obj}T_{base} $$

The satellite tracking program uses two frame buffers (0 and 1) to acquire a new stereo pair of images every 1/30 second. Image acquisition is controlled by a KW11-P interrupt routine as in the correlation tracker. The processing of each image consists of thresholding three search windows centered at the predicted label locations. The image coordinates of "white" pixels are summed ($\sum I$, $\sum J$) to calculate the centroid of the label. Corresponding centroids from the two images are passed to INTSCT to obtain the 3-D location of the label. The 3-D measurements are smoothed by a third-order recursive filter. The smoothed coordinates are then used to calculate ${}^{obj}T_{base}$.

Operator interaction for initialization consists of entering a value for the filter characteristic time constant (typically 7) and designating the initial label locations in each image with a cursor.

7.3 3-D Object Tracker

The 3-D object tracking program TRACKOBJ tracks known

objects represented by wire-frame models [8]. The output of the tracker is a complete description of the position and velocity of the object -- three linear and three angular components. In its present form, the tracker only models convex polyhedra. The modelling capability will be extended to include concave polyhedra and objects with cylindrical surfaces.

The tracking algorithm consists of four steps: PREDICTION, PROJECTION, MEASUREMENT, and ADJUSTMENT. A simplified description of each step is given below:

1) PREDICTION - noting the elapsed time between the current and previous images, the expected position and orientation of the object are calculated.

2) PROJECTION - visible edges of the object are projected into the image plane.

3) MEASUREMENT - the IMFEX edge array is searched to locate edge elements in the vicinity of the projected edges. The discrepancy between observed and predicted edge locations is recorded each time an edge is found.

4) ADJUSTMENT - edge location discrepancies are processed by a least squares adjustment to generate new position and velocity measurements.

Important features of the tracking algorithm include

1) significant noise is tolerated.

2) missing edges, due either to poor contrast or occlusion can be tolerated.

3) the algorithm will work with any number of cameras including only one.

The program is initialized by the operator who enters the initial position and orientation of the object. A menu lists other parameters that can be modified including the number of cameras to use, the edge threshold to set in IMFEX, the object model file name specification, and the values of tuning parameters. A menu of parameters is displayed to aid the operator. All parameters have default values.

7.4 Acquisition

The acquisition program finds a known moving object to initialize the object tracking program described in the previous section [9]. Together, the acquisition and tracking programs form a system capable of detecting and tracking known 3-D objects.

The acquisition algorithm represents a special case from the general domain of 3-D object recognition, and is distinguished from the latter primarily by the fact that the identity of the object is known a priori. Also the acquisition algorithm relies on motion cues to extract 3-D information (in fact the object must be moving for the acquisition algorithm to work).

As shown in Figure 7-2, the acquisition program consists of five modules which are partitioned to run as two concurrent processes. One process consists of the vertex tracker which extracts vertices from the binary edge output of IMFEX and continuously updates the 2-D (image) position and velocity of detected vertices. The other process consists of the remaining four modules which do the 3-D analysis resulting in an estimate of the object's position, orientation and velocity which are used

to initialize the object tracker. Notice that data from the vertex tracker enters the other process at two points. The input to the motion stereo module represents the initial input to the 3-D analysis process. Several seconds elapse before a model match is obtained with these data. Thus the tracking initializer needs the latest information from the vertex tracker to obtain a current estimate of the object's position and velocity. A brief description of each module is given below.

VERTEX TRACKER: Vertices are represented as 5 x 5 binary edge patterns. The algorithm effectively does a table-lookup based on converting the 5 x 5 pattern into a 25-bit lookup table address. To avoid the prohibitive memory requirements of a 2^{25}-element table, the actual implementation uses a two-stage table-lookup scheme based on overlapping 3x3 patterns which fit in a 5x5 window. The vertex tracker is interrupt driven to run at 60 Hz. When multiple cameras are being used, the tracker switches to a new camera at the beginning of each iteration. 2-D position and velocity information is stored in a shared memory segment for direct access by the motion stereo and tracking initializer modules.

MOTION STEREO: The motion stereo module accepts 2-D vertex position and velocity information from the vertex tracker. Using a rigid body assumption, the 3-D position of each vertex is calculated. These measurements are assumed to have bias and scale factor uncertainties.

STEREO MATCHER: The stereo matcher matches vertices between the two cameras to obtain accurate 3-D position measurements. As a result, scale factor and bias uncertainties are reduced so that the position estimates for unmatched vertices are also improved.

MODEL MATCHER: Computed vertex positions are compared to the object model to determine the position and orientation of the object.

TRACKING INITIALIZER: Using the latest 2-D vertex positions from the vertex tracker, this module performs a weighted least squares adjustment of the object position, orientation and velocity. The result is passed to the object tracker.

Each of the 3-D analysis modules has a success/failure criterion. If a module fails, control is transferred back to the motion stereo module to make a new attempt. Likewise, if the object tracker fails, the acquisition program is started again to reacquire the object.

THE ACQUISITION PROGRAM

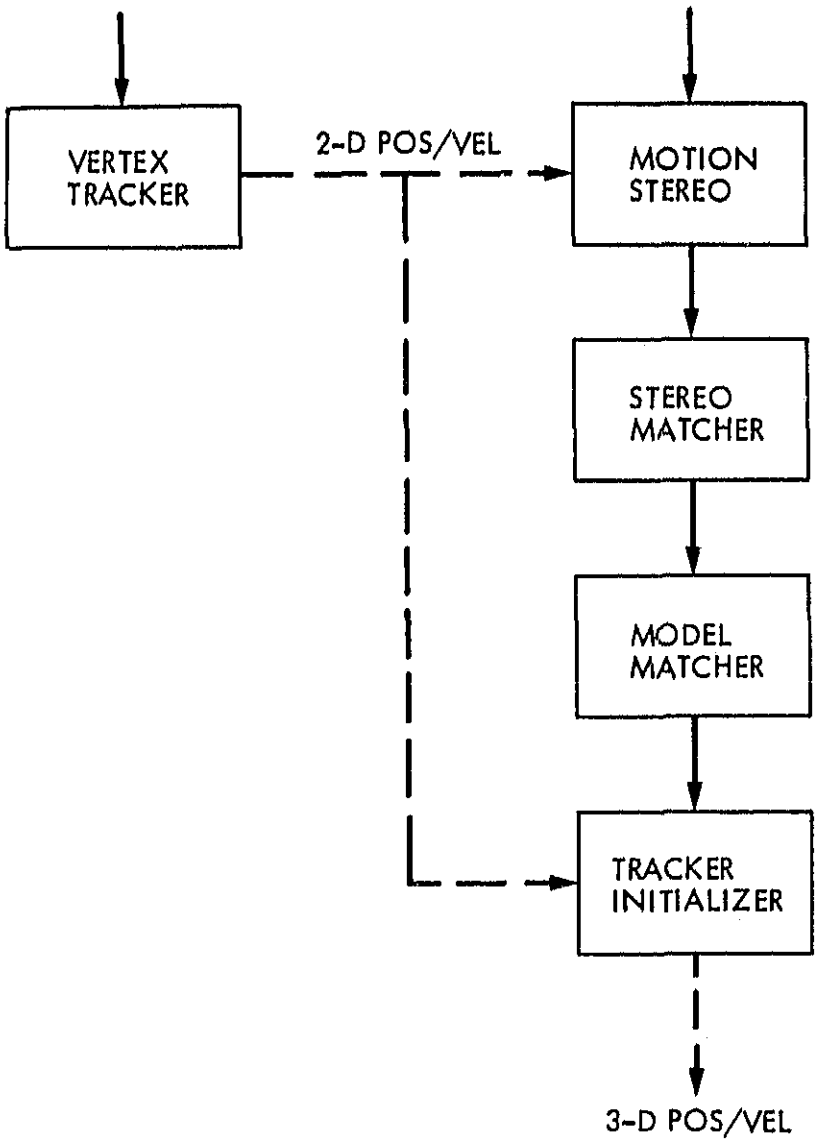


Figure 7-2. Control and Data Flow of Acquisition System

REFERENCES

1. R. Eskenazi and J. M. Wilf, "Low-Level Processing for Real-Time Image Analysis," JPL Pub. 79-79, Pasadena, CA, 1979.

2. C. A. Rosen and D. Nitzan, et al., "Exploratory Research in Advanced Automation," Fourth Report, SRI International, Menlo Park, CA, 1975.

3. Y. Yakimovsky and R. Cunningham, "On the Problem of Embedding Picture Elements in Regions," TM 33-774, Jet Propulsion Laboratory, Pasadena, CA, 1976.

4. Y. Yakimovsky and R. Cunningham, " A System for Extracting Three-Dimensional Measurements from a Stereo Pair of TV Cameras," Computer Graphics and Image Processing, Vol. 7, pp 195-210, 1978.

5. M. D. Griffin, R. T. Cunningham, and R. Eskenazi, "Vision-Based Guidance for an Automated Roving Vehicle," Paper 78-1294, AIAA Guidance and Control Conference, Aug. 7-9, 1978, Palo Alto, CA.

6. T. Brooks, "The Mathematics of Supervisory Computer Control of Manipulators," Journal of Dynamic Systems, Measurement & Control, Vol. 104, Dec. 1982, pp. 323-330.

7. R. Cunningham and T. Brooks, "Vision-Based Real-Time Robot Control for Capturing Moving Objects," to appear in 1984 American Control Conference, San Diego, CA, June 6-8, 1984.

8. D. Gennery, "Tracking Known Three-Dimensional Objects," Proceedings of the AAAI Conference, August, 1982, Pittsburgh, PA. pp. 13-17.

9. D. Gennery, "Stereo Vision for the Acquisition and Tracking of Moving Three-Dimensional Objects," Proceedings of the AAAI Workshop on Sensors and Algorithms for 3-D Machine Perception, Aug. 1983.