NASA
National Aeronautics and
Space Administration

JSC-24630

Lyndon B. Johnson Space Center
Houston, Texas 77058

VISION TRACKING SYSTEM FOR THE EXTRAVEHICULAR ACTIVITY RETRIEVER ROBOT

Contract NAS 9-17900

Job Orders 16-307, 16-E01

March 1991

Prepared by Lockheed Engineering & Sciences Company,
Houston, Texas, for the Tracking and Communications
Division of the Engineering Directorate

Lockheed
Engineering & Sciences Company

LESC-28763

Lockheed
Engineering & Sciences Company

ENGINEERING AND SCIENCE PROGRAM
2400 NASA Road 1, P. O. Box 58561, Houston, Texas 77258 (713) 333-5411

March 26, 1991
MEMO: 91-089

TO: J. C. Lamoreux
NASA/JSC, EE6

FROM: J. W. Teague
Lockheed, C22

SUBJECT: TRANSMITTAL MEMO FOR THE TECHNICAL REPORT ENTITLED, "VISION TRACKING SYSTEM FOR THE EXTRAVEHICULAR ACTIVITY RETRIEVER ROBOT", LESC-28763, JSC-24630, MARCH, 1991

REF: Contract No. NAS9-17900
Job Order Nos. 16-307 and 16-E01

The attached technical report prepared by Alan Smith and Sunil Fotedar provides a description of the algorithm for target identification and tracking of three defined targets used in the second phase demonstration of the Extravehicular Activity Retriever (EVAR) robot. The software development for the communications link between JSC buildings 14 and 9A and the target identification and tracking algorithm implementation using the APA512s machine vision system are described.

James W. Teague

J. W. Teague
Operations Manager
Tracking and Communications
Department

Attachment

R. A. Tremant, EE w/o attachment
J. C. Lamoreux, EE6
H. A. Nitschke, EE6
T. E. Fisher, EE6
R. D. Juday, EE6
D. E. Rhoades, ER331
R. Goode, ER231
K. A. Grimm, ER421
K. H. Vorhaben, C02
S. E. Monroe, C02
J. M. Victor, C02
A. T. Smith, C02, (2)
S. Fotedar, MDC B2MC (2)
J. O. Files
Library (5)

CONTENTS

TABLES

FIGURES

ACRONYMS AND ABBREVIATIONS

APBF	Air Precision Bearing Floor
CCD	charge coupled device
EVA	extravehicular activity
EVAR	EVA Retriever
FOV	field of view
I/O	input/output
JSC	Lyndon B. Johnson Space Center
McDAC	McDonnell Douglas Astronautics Company
ORU	orbital replacement unit
SSF	Space Station Freedom
TCD	Tracking and Communications Division
TTB	Tracking Techniques Branch
VFIR	video finite impulse response
VME	Versa Module Eurocard
XOR	exclusive OR

1. SUMMARY

This report describes the application of the Adaptive Automation, Inc., machine vision processor, model APA512S, for vision tracking support to the Extravehicular Activity (EVA) Retriever (EVAR) robot. This vision tracking system provides performance superior to the McDonnell Douglas Astronautics Company (McDAC) vision tracking system presently being used.

Target recognition, target tracking, and input/output (I/O) communications interface software are presented. Target recognition is accomplished with an artificial neural network. The communications interface software sends the processed vision information from the APA512S, located in building 14 at the Lyndon B. Johnson Space Center (JSC), to the central EVAR data collection point, the vision subsystem (VISION_CTL), located in building 9A.

2. INTRODUCTION

The function of the EVAR robot, being developed at JSC, is to retrieve free-flying objects in space. During the development and operation of Space Station Freedom (SSF), EVA crewmen will have the task of assembling and maintaining the Station and platforms (ref. 1). It is possible that an EVA crewman or a tool will separate from SSF during EVA operations. Since it may not be practical, because of safety or time considerations, for a crewman to leave the Station and rescue the person or object, the retriever is designed to provide this capability.

The development of the EVAR is partitioned into three ground demonstration phases: phases I, II, and III. Each phase requires additional complexity in the hardware and software design. In the phase I demonstration the robot retrieved a stationary object with no obstacles in its path. It had only one video camera and a three-dimensional Odetics laser mapper that acted as sensors. The McDAC machine vision system, the McDAC Tracker, provided the image processing for this phase. The retriever is currently being prepared for the phase II demonstration, in which the robot will encounter stationary obstacles en route to its target. The robot is equipped with six video cameras, positioned at various locations on the robot body, in addition to the three-dimensional laser mapper. A newer and more sophisticated system is needed for this demonstration. Because of the age of the McDAC Tracker, it continually needs repair and its technology is outdated. The APA512 and Datacube boards were purchased to provide the robot with the additional "intelligence" needed. Section 3.2 gives a brief description of the APA512 and Datacube boards. In phase III the robot will encounter moving obstacles in its path.

Figure 2-1 is a photograph of the robot in the phase I demonstration configuration. Some modifications have been made to the exterior of the robot since this photo was taken. The head of the EVAR is now on a turntable that allows 360° of movement. Also, three chest cameras are used instead of the one shown.

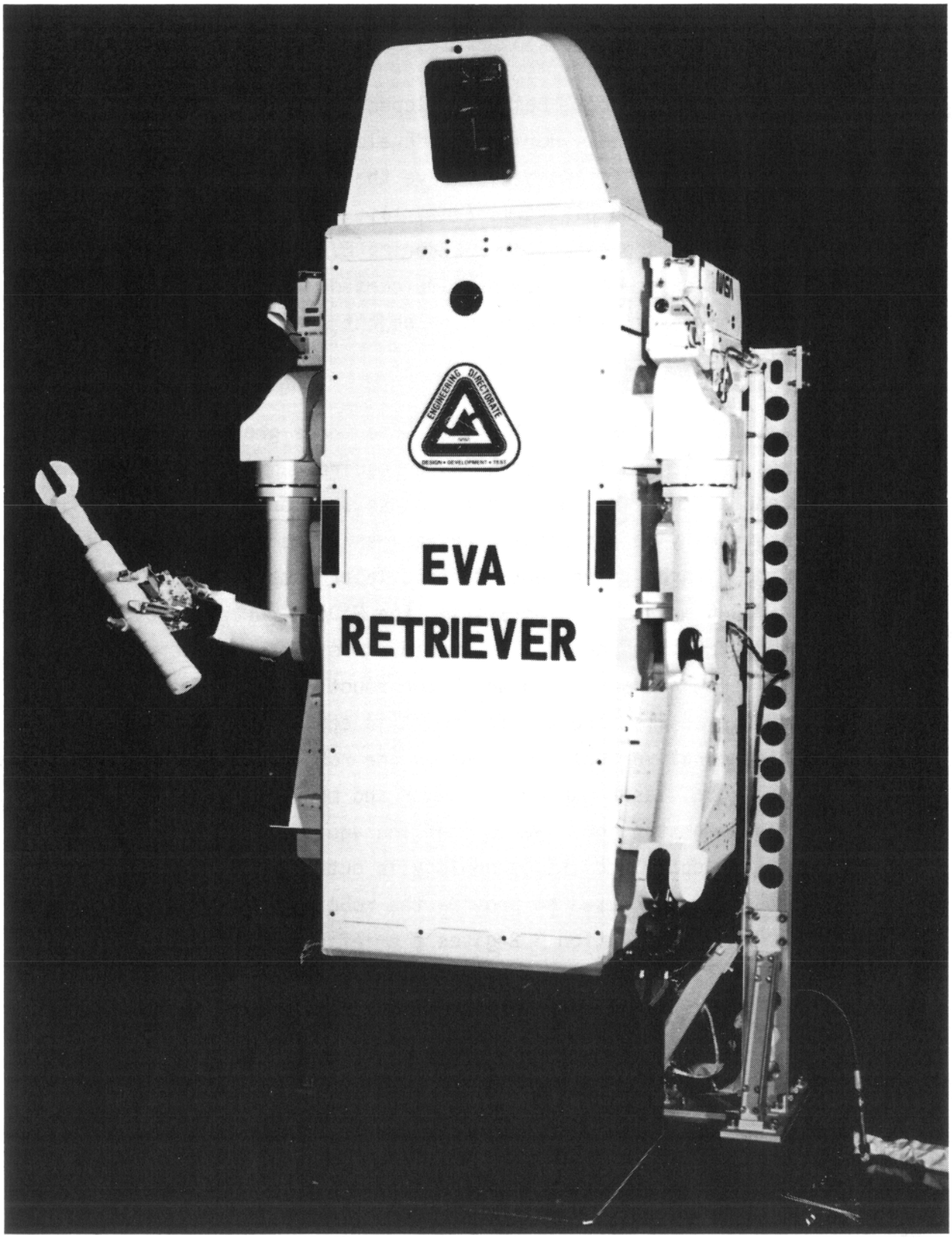

Figure 2-1.- EVAR, phase I demonstration.

The goal of the Robotic Vision/Tracking Sensors Project is to replace the McDAC Tracker with the APA512S machine vision processor and provide vision support for the EVAR. Section 3 gives a description of the software implementation of the routines outlined in the EVAR phase II software interface requirements manual (ref. 2). A review of that document is encouraged to gain a familiarization with the control and information interface between the APA512S and the vision subsystem of the EVAR, VISION_CTL. A functional description of the APA512 and Datacube boards is given in section 3. A more detailed analysis can be found in references 3 through 6. Also, the neural network algorithm used for target identification is briefly discussed. Planned features of the software that are beyond the requirements of the phase II demonstration, such as target tracking with occlusions, are not discussed, because they are in the development stages.

The I/O serial link interface concepts are given in section 4. The commands requesting vision-related tasks for the EVAR are received over an optical fiber link from the vision subsystem of the EVAR in the form of a data stream. The EVAR is located in JSC building 9A, and the APA512S is in building 14. The communication software interprets the message and responds to it after the appropriate processing is performed.

Conclusions are given in section 5, and references are listed in section 6. Appendix A gives the software documentation of the routines used to accomplish the EVAR vision task. The source code, written in C language, is given in appendix B.

3. EXTRAVEHICULAR ACTIVITY RETRIEVER VISION PROCESSING

3.1 OVERVIEW OF THE APA512

The APA512 is a hardware implementation of a two-dimensional image feature extractor (ref. 7). The input to the APA512 is binary digitized video representing white "blobs" on a black background. The features returned are area parameters describing the blobs in the camera's field of view (FOV).

The area parameters in table 3-1 are divided into four groups: hardware-calculated seed parameters, location dependent parameters, orientation dependent parameters, and discrimination parameters. Table 3-1 lists the most useful of the 53 total parameters available. The 15 seed parameters are calculated by the APA512 hardware at the real-time video frame rate. All other parameters are derived from the seed parameters. With the use of a mixture of these parameters, the APA512 is programmed to interpret the video information provided by the EVAR's cameras. These parameters are defined in table 3-2.

3.2 DESCRIPTION OF THE MACHINE VISION PROCESSING SYSTEM

The APA512 is configured as a Versa Module Eurocard bus (VMEbus) slave interfaced to a Datacube MAXBUS. In this report the term "APA512" refers to the actual hardware boards, and "APA512S" refers to the combination of the APA512 and the Datacube boards. As shown in figure 3-1, three Datacube boards serve as preprocessors for the APA512. The DIGIMAX board digitizes RS170 format source video from JSC building 9A. The video finite impulse response (VFIR) board convolves the digitized image with the 3×3 blurring kernel {1,1,1,1,0,1,1,1,1} to eliminate spatial discontinuities in the blob due to shadows (ref. 8). Thresholding of the 8-bit grey level data is accomplished by the MAX-SP board. The APA512 processes binary images only. VISION_CTL adjusts the level via the SET_THRESHOLD command (described in section 3.4.7). The MAXGRAPH board is used to display boxes and/or cross hairs overlayed on selected blobs of the video data.

TABLE 3-1.- APA512 AREA PARAMETERS

(a) HARDWARE-CALCULATED SEED PARAMETERS

Parameter	Description
sumx	Sum of the x coordinates of all pixels in the blob
sumy	Sum of the y coordinates of all pixels in the blob
sumxx	Sum of the squares of the x coordinates in the blob
sumyy	Sum of the squares of the y coordinates in the blob
sumxy	Sum of the products of x and y coordinates in the blob
npixels	Number of pixels in the blob
perimx	x coordinate of the last point on the blob's perimeter
perimy	y coordinate of the last point on the blob's perimeter
xmin	Minimum x coordinate of the blob
xmax	Maximum x coordinate of the blob
ymin	Minimum y coordinate of the blob
ymax	Maximum y coordinate of the blob
perimeter	Length of the perimeter of the blob
polarity	Background or foreground indicator
border	Frame border intersection indicator

(b) LOCATION DEPENDENT PARAMETERS

Parameter	Description
xcent	x coordinate of the centroid
ycent	y coordinate of the centroid

TABLE 3-1.- APA512 AREA PARAMETERS (Concluded)

(c) ORIENTATION DEPENDENT PARAMETERS

Parameter	Description
angle	Angle of statistically equivalent ellipse
xdiff	Length of blob in x direction
ydiff	Length of blob in y direction

(d) DISCRIMINATION PARAMETERS

Parameter	Description
major	Major axis length of statistically equivalent ellipse
minor	Minor axis length of statistically equivalent ellipse
axratio	*minor/major*
peround	$npixels/perimeter^2$

TABLE 3-2.- PARAMETERS USED FOR EVAR VISION

Parameter	Description
area	Size of the blob, related to number of pixels
(xcent,ycent)	(x centroid,y centroid)
maxx	Largest x coordinate of the blob
minx	Smallest x coordinate of the blob
maxy	Largest y coordinate of the blob
miny	Smallest y coordinate of the blob
xdiff	*maxx – minx* (width of the blob)
ydiff	*maxy – miny* (height of the blob)
axratio	(Minor axis length)/(major axis length) of statistically equivalent ellipse
peround	$area/perimeter^2$
edge_polarity	Gives color and determines if the blob is fully in the camera's FOV

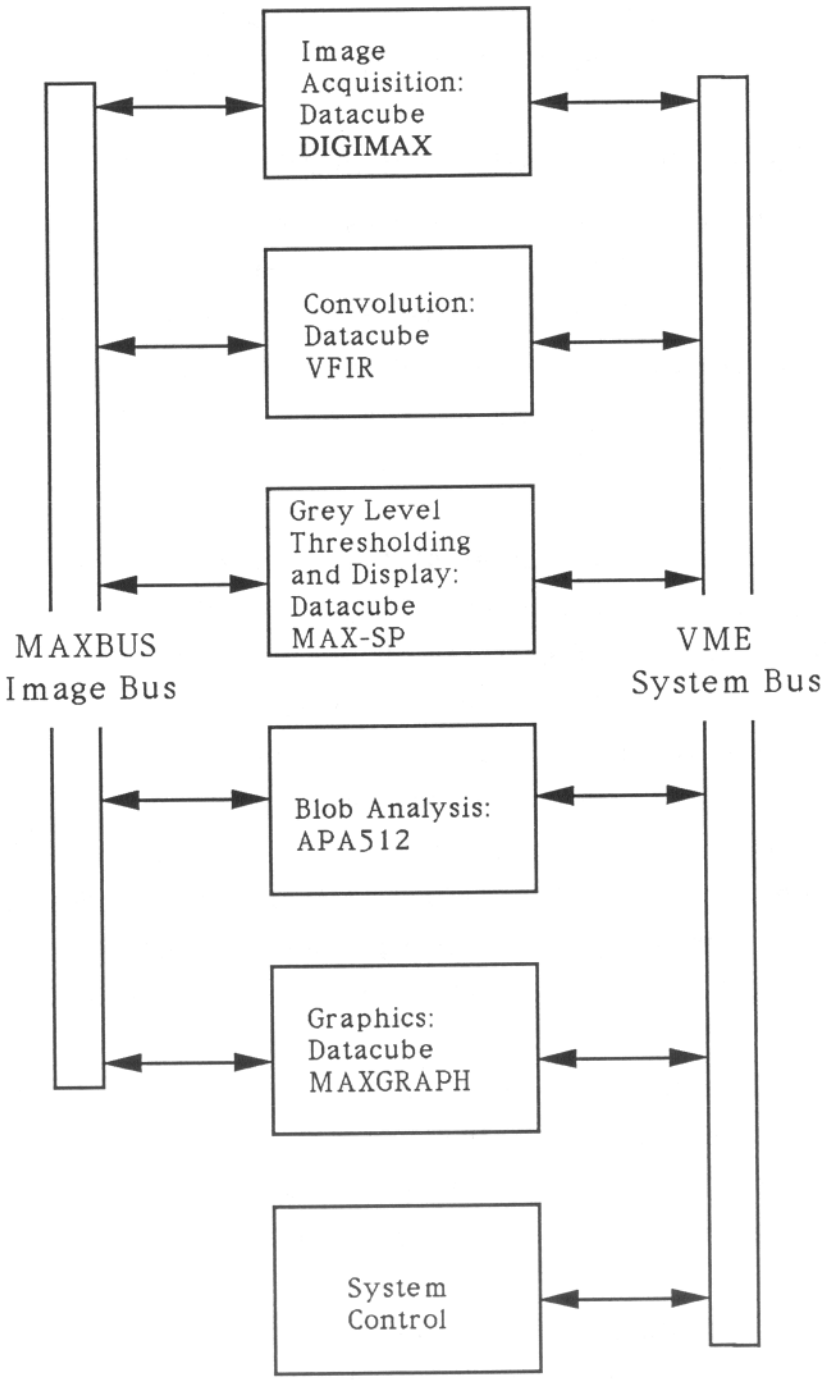

Figure 3-1.- Image processing system configuration.

Of the 53 blob parameters available from the APA512, the 12 used for image processing are listed and briefly described in table 3-2. *xcent*, *ycent*, *xdiff*, and *ydiff* are among the data sent to VISION_CTL for target/obstacle verification and robot path planning. The rest of the parameters are used for pattern recognition and are known internally to the APA512 only.

3.3 PATTERN RECOGNITION WITH ARTIFICIAL NEURAL NETWORKS

An artificial neural network was chosen as a means for pattern recognition because it was determined to be superior to conventional statistical techniques in its classification capability. The neural network not only makes decisions more quickly, it also generalizes better from data not in the training set. In other words the network more accurately classifies samples "close" to the data in the training set.

The goal of any classifier, whether statistical or neural network, is to classify $\mathbf{N}$ objects into $\mathbf{M}$ classes. Consider first a statistical classifier, such as the Bayesian classifier (refs. 9 and 10). In figure 3-2, symbols (object features) are input into the system so that a matching score can be calculated for each of the $\mathbf{M}$ classes. The score is computed based on parameters previously estimated in the training process. If the input symbols are assumed to be distributed normally, the statistical parameters will be the mean vector $\mathbf{v}$ and covariance matrix Σ of the features. Methods and equations for computing the matching score will not be discussed here; references 9 and 10 contain the background, if it is needed.

Each object has $\mathbf{M}$ matching scores, and the largest is chosen to determine to which class the object belongs. Thus, $\mathbf{M} \times \mathbf{N}$ calculations must be made. It is also worth noting that the success of the statistical classifier is limited by the accuracy of the parameters $\mathbf{v}$ and Σ.

Second, consider the neural network classifier of figure 3-3. The network has an input layer, a middle layer of neurons, and an output layer of $\mathbf{M}$

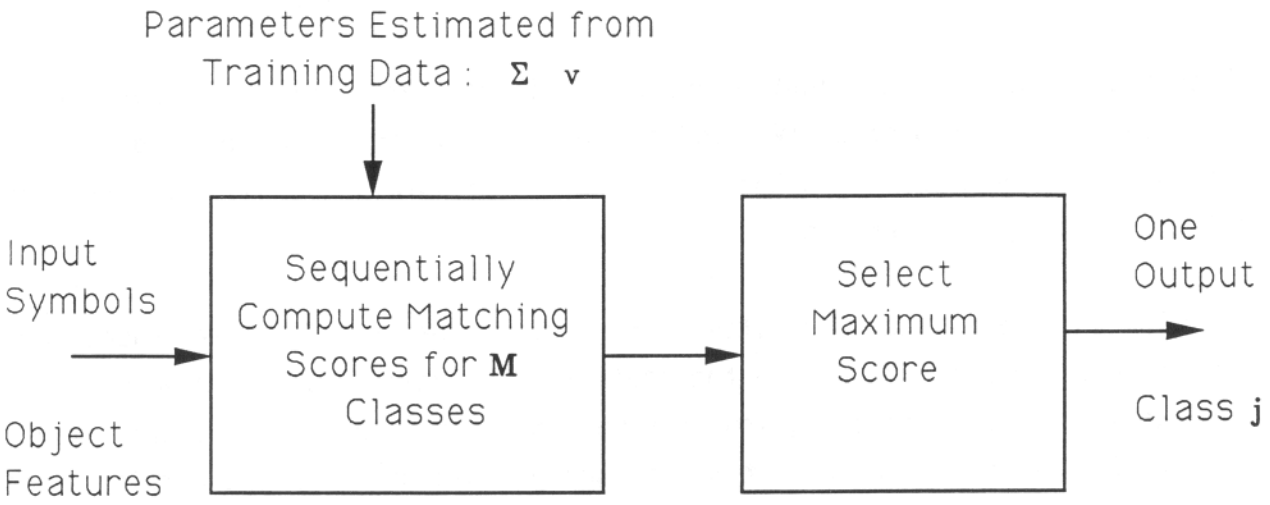

Figure 3-2.- Operation of statistical classifier.

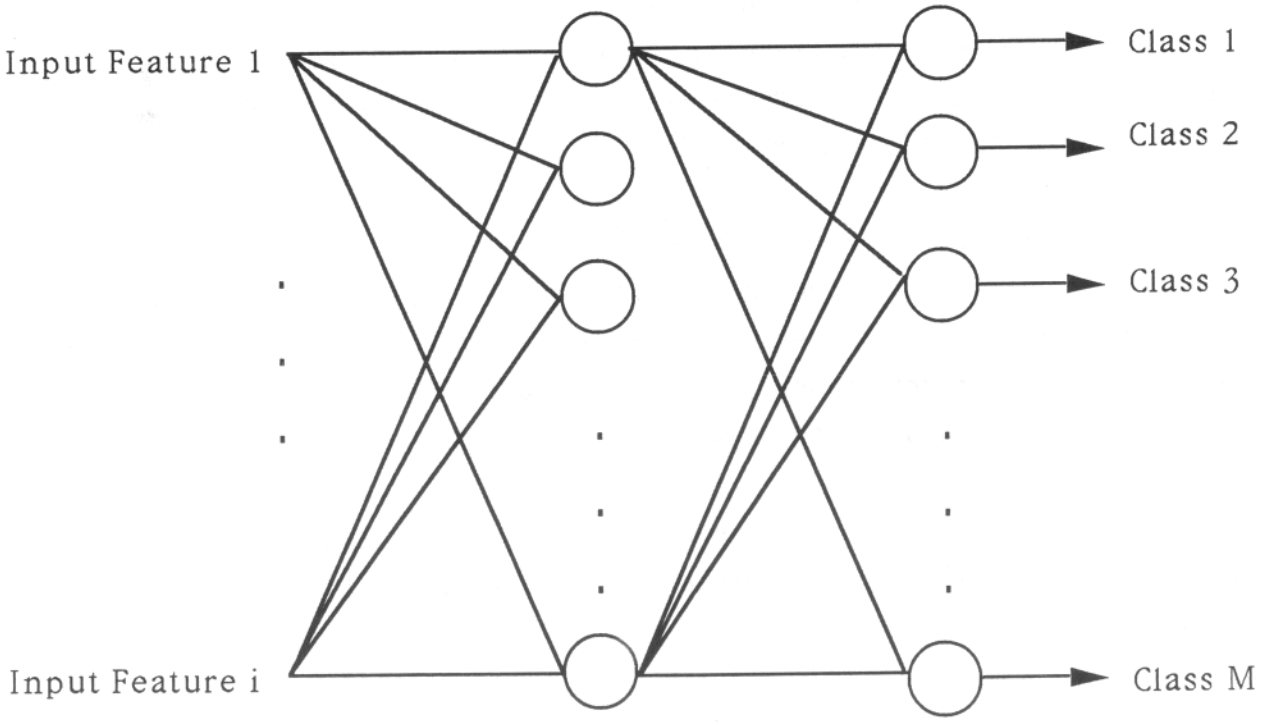

Figure 3-3.- Neural network used for pattern recognition.

neurons. Each layer of neurons is fully connected to the next layer, except for the output layer. Each object feature has an input node. The state of each output neuron is the matching score for the *jth* class.

Each neuron performs a weighted sum of its inputs, as shown in figure 3-4. The weights are denoted as W_i, and the inputs from the previous stage are X_i. The sum is passed through a hard-limiter, f_h, to obtain the output. The value of the output is either 0 or 1.

As with the statistical classifier, the symbols for one of the N objects are input into the network; unlike the statistical classifier, the network computes all the M matching scores simultaneously. Thus, the network is at least M times faster at computing the scores, depending on the complexity of the rule for computing the scores in the statistical classifier.

Another area of superiority for the neural network is classification accuracy. In training, the network makes no errors in modeling the data distribution parameters, such as v and Σ, because no assumptions about the distribution form are made. The network adaptively calculates its own data distribution based on the training samples. Thus, the user need only make certain that the training samples are representative of the entire set of input symbols that the network will see.

The network was trained to recognize three classes of objects: astronauts, wrenches, and orbital replacement units (ORU's). *axratio* and *peround* are the two input features to the network. These features were chosen because they are invariant to translation, scale, and rotation parallel to the camera plane. Thus, the goal is two-dimensional recognition.

The weights were adapted with the well-known back propagation learning algorithm (ref. 11) and were initially set at random values. Training is accomplished by presenting the network with *axratio* and *peround* data, calculating the output, and updating the weights according to the algorithm. For example, if astronaut *axratio* and *peround* values are given to the network

as inputs, the network should have an output of {1,0,0}. Any other output will cause the weights to be changed.

Once the network is trained, its function is to output {1,0,0} when given astronaut data, {0,1,0} when given wrench data, and {0,0,1} when given ORU data. Figure 3-5 shows the training data points and decision regions chosen by the network. Areas that are white indicate that no decision was made. In these cases all outputs are 0, or more than one output is 1. From this figure it is known how the network will classify given any *(axratio,peround)* data point.

As the network trains, the three regions for ASTRONAUT, WRENCH, and ORU classifications shrink around the training points. If the network is trained too long, the regions will include only the training points and the network cannot generalize as well. On the other hand, if the training is not long enough, too many "false alarms" will occur. Normally, the network is trained and tested in cycles until satisfactory results are achieved. Thus, the length of training depends on the false alarm rate that is tolerable. In this case the network could have been trained longer to improve its ability to reject clutter.

3.4 VISION PROCESSING COMMANDS

The commands to which the APA512S must respond from VISION_CTL can be divided into three groups: utility routines, target recognition routines, and the obstacle/target tracking routine. The commands composing each group are listed in table 3-3. All macros are defined in the header file named *vision.h*.

3.4.1 ENABLE

The ENABLE command activates the hardware connection between the Datacube and the APA512 boards. A software structure channel is established to read the data from the APA512 board.

TABLE 3-3.- COMMANDS FROM VISION_CTL

Command group	Commands
Utility routines	ENABLE DISABLE RESET ARE_YOU_HERE_TODAY SEND_HEALTH_STATUS SET_TIME SET_THRESHOLD STOP_BLOB_TRACK B14_VIDEO_SELECT_COMPLETE
Target recognition routines	TARGET_ACQ_MANUAL TARGET_ACQ_AUTO BLOB_DETECT DETERMINE_GRASP MONITOR_GRASP
Obstacle/target tracking routine	BLOB_TRACK

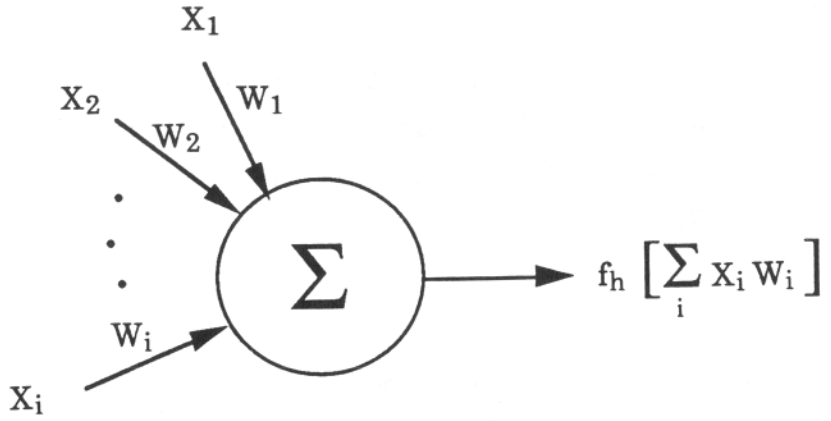

Figure 3-4.- Function of each neuron.

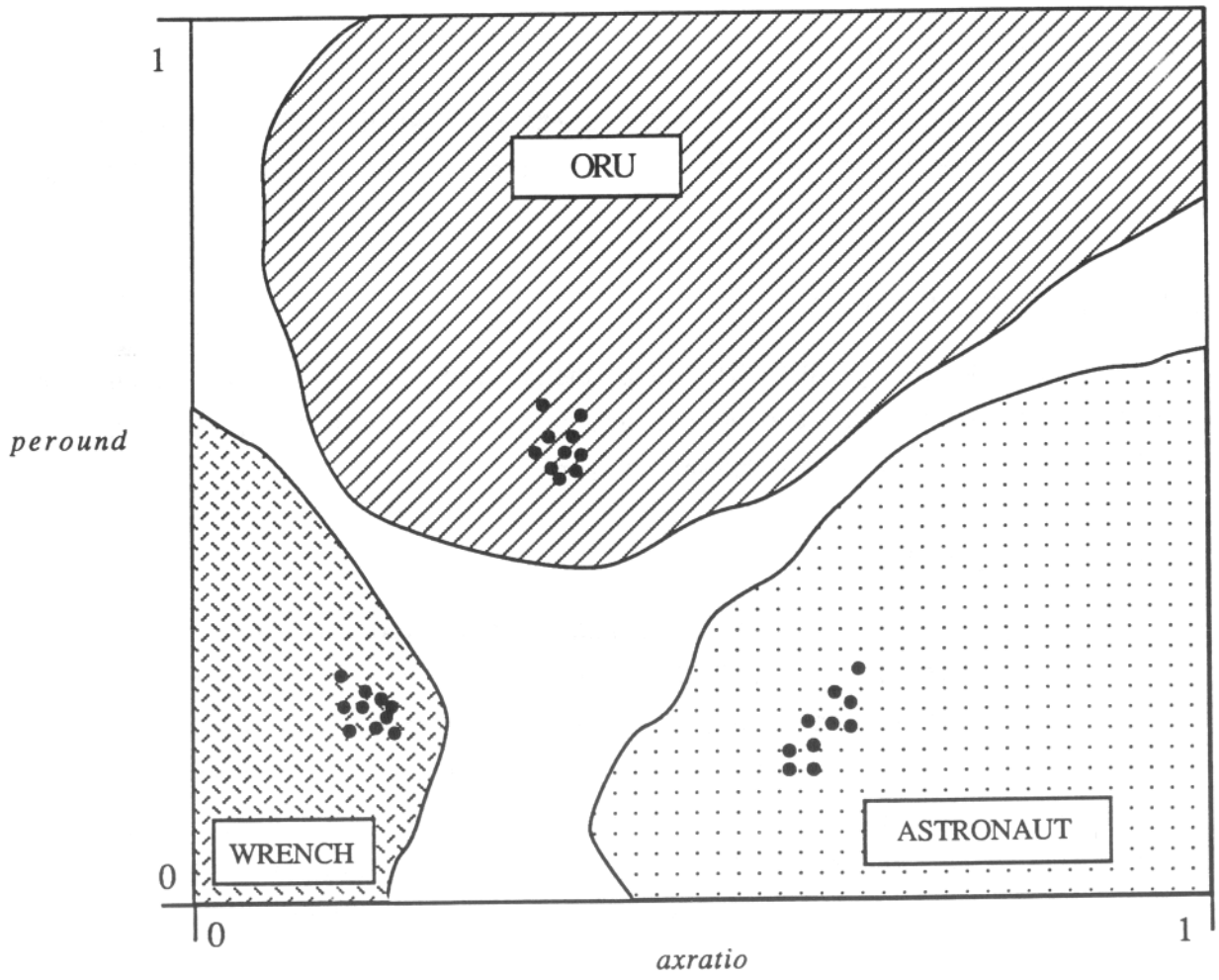

Figure 3-5.- Decision space of the network for *(axratio,peround)* values.

3.4.2 DISABLE

A DISABLE command first frees the allocated data arrays and internal structures to the APA512S and then disconnects the board configuration. Before the boards can be enabled again, the initialization sequence "... ?!" must be read from VISION_CTL.

3.4.3 RESET

In case of a hardware error read from one of the processing boards, VISION_CTL may wish to reestablish the software data channel and hardware connection with the RESET command. For the APA512S, RESET and DISABLE perform the same function since the initialization sequence must be read before the boards can be enabled.

3.4.4 ARE_YOU_HERE_TODAY

In response to the ARE_YOU_HERE_TODAY command, I_AM_HERE_TODAY is sent to VISION_CTL if the boards are enabled successfully; otherwise, NOT_HERE is sent.

3.4.5 SEND_HEALTH_STATUS

The reply to the SEND_HEALTH_STATUS command also depends on the success of enabling the boards. If the boards are enabled successfully, NOMINAL is sent to VISION_CTL; otherwise, ABNORMAL is sent.

3.4.6 SET_TIME **modifier**

The EVAR phase II software interface requirements manual (ref. 2) requires each data structure sent to VISION_CTL to have a time stamp that records when the raw video data was acquired. The SET_TIME command synchronizes the APA512S and other EVAR subsystems with VISION_CTL. The modifier of this command defines the time for the entire system. When SET_TIME is received, the difference between the modifier and the APA512S operating system time is taken so that the correct time stamp on the raw video data can be calculated.

3.4.7 SET_THRESHOLD **modifier**

A dynamic APA512S threshold is available with the SET_THRESHOLD command. The desired value is the unsigned integer modifier ranging from 0x00 to 0xff, where 0x00 is black and 0xff is white.

3.4.8 STOP_BLOB_TRACK

The STOP_BLOB_TRACK command received from VISION_CTL allows the APA512S to suspend one of the three blob-tracking routines (TARGET_ACQ_AUTO, BLOB_DETECT, or BLOB_TRACK) so that it may receive another command from VISION_CTL.

3.4.9 B14_VIDEO_SELECT_COMPLETE

The B14_VIDEO_SELECT_COMPLETE command is the response from VISION_CTL after B14_VIDEO_SWITCH_REQUEST has been sent from the APA512S. B14_VIDEO_SWITCH_REQUEST is always sent before any blob-tracking routine is invoked in order to obtain video from a particular camera.

3.4.10 TARGET_ACQ_MANUAL **modifier**

The function of the APA512 for the TARGET_ACQ_MANUAL command is to return information about each target/obstacle in the FOV of camera **modifier.** The long form of the data structure listed in the interface manual (ref. 2) is "filled" with the appropriate values calculated by the APA512S. This routine is considered "manual" because the McDAC Tracker did not perform recognition; VISION_CTL would manually choose which blobs were the obstacles/targets. With the APA512S and neural network recognition scheme, however, the title for this command is no longer accurate.

Figure 3-6 is a functional block diagram depicting the TARGET_ACQ_MANUAL process. The APA512S initially requests video from VISION_CTL for the appropriate camera. Upon successful completion of the switch, the interpretation of the scene has begun. The blobs in the FOV were identified as being either targets or obstacles.

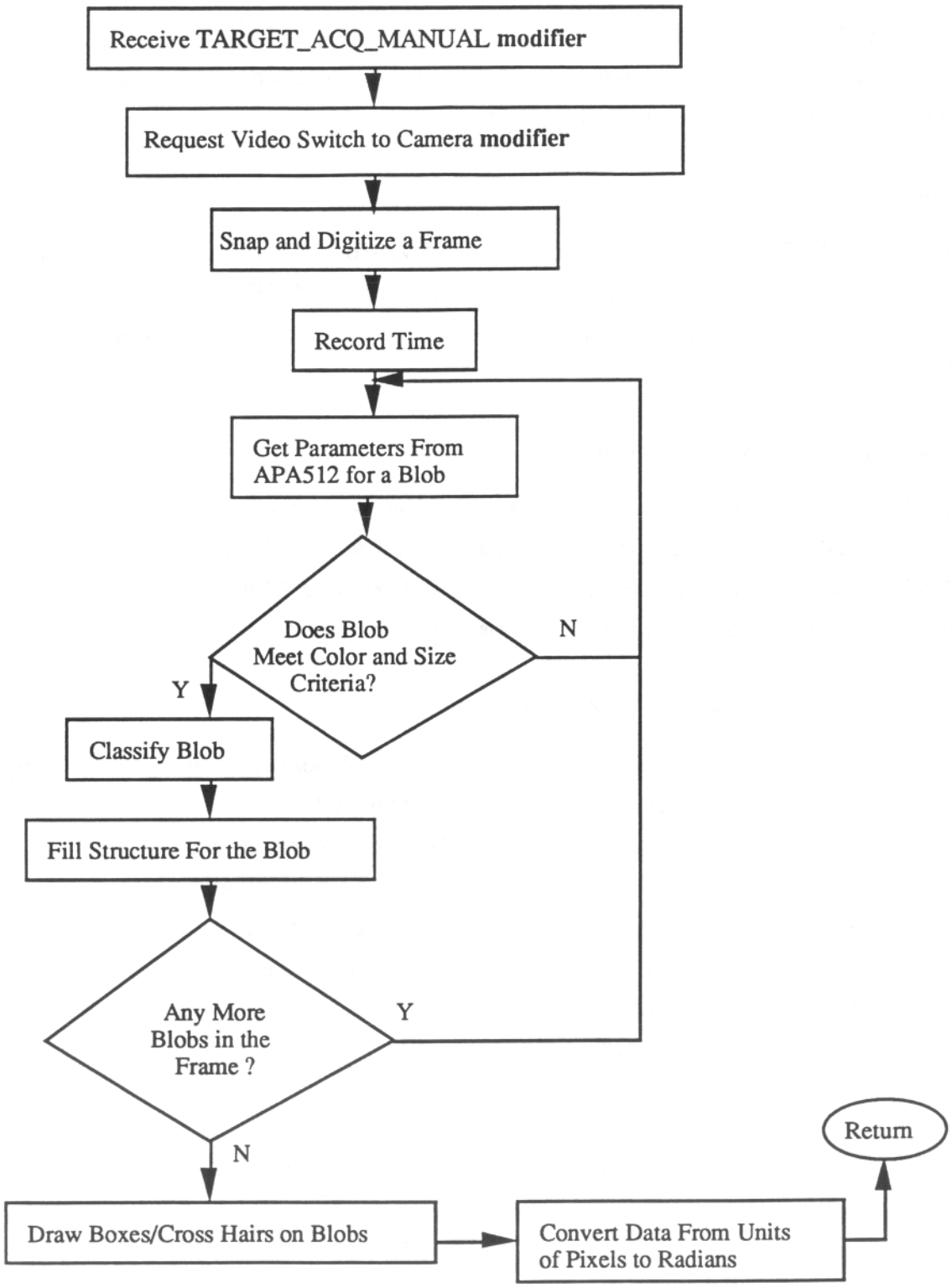

Figure 3-6.- TARGET_ACQ_MANUAL functional block diagram.

Not all blobs in the video frame are reported; the APA512S filters blobs too small or large with the parameter *area*. The *vision.h* macros, MAX_SIZE and MIN_SIZE, define the acceptable range. Also, *edge_polarity* allows only white blobs to be processed.

The blobs passing these constraints are classified with the neural network. If a blob belongs to one of the three classes, the "idclass" structure member is filled as IDCLASS_KNOWN; otherwise, it is filled as IDCLASS_OBSTACLE. The "idcode" is ASTRONAUT, WRENCH, ORU, or NEITHER, depending on the network's decision.

"quality" is a measure of how well the APA512 *(axratio,peround)* data fits the training set, and is between 0 and 1. The members "blobnum" and "lblobnum" are assigned to the blob number defined during the video scanning process. "procid" and "timestamp" are the APA512S process identification number and the time the video was acquired, respectively.

The remaining structure members describe the physical qualities of the blobs. Notice that all y-coordinate data must be "stretched" to compensate for the fact that the APA512 processes only one of the two fields that make the video frame. SCALE_YCENT accomplishes this task. Any structure member not being computed for the phase II demonstration is defined as NULL_DATA_FLOAT.

Graphic overlays of the raw video data are accomplished with the Datacube MAXGRAPH board. Blobs in the ASTRONAUT class are bounded with a box, and wrenches are designated with cross hairs. The ORU class is noted with both.

Finally, all blob measurements must be converted from pixels to radians before being sent to VISION_CTL. The origin must be shifted from the camera's upper left corner to the middle of the FOV by subtracting RES_X/2.0 and RES_Y/2.0 from all x and y coordinates, respectively. These macros define the pixel resolution of the DIGIMAX. Next the data must be transformed to radians. The camera FOV, FOV_X and FOV_Y, is 46.2° and 36.2° for the Elmo EM-102 charge coupled device (CCD) color video camera. The

pixel-to-radian conversion factor is derived by changing the angles to radians and dividing by the appropriate resolution.

NO_BLOBS_FOUND is returned to VISION_CTL if the number of blobs is 0 or greater than 255. The APA512 hardware is limited to processing a maximum of 255 blobs.

3.4.11 TARGET_ACQ_AUTO **modifier**

In the TARGET_ACQ_AUTO command, the modifier is the object class to recognize. There can be more than one object in each class. The goal is to sequence through each of the six cameras one time and return information on blobs belonging to the requested class. Information on blobs that belong to other classes (besides class UNKNOWN) will be reported, as well, but if no blob can be found belonging to the class **modifier**, TARGET_NOT_IN_SCENE will be sent to VISION_CTL.

The block diagram of figure 3-7 is very similar to the one of figure 3-6. The main difference is that the APA512S does not send VISION_CTL information about obstacles with TARGET_ACQ_AUTO.

3.4.12 BLOB_DETECT

The BLOB_DETECT command is an extension of TARGET_ACQ_AUTO that puts the APA512S in a continuous pattern recognition mode. Instead of stopping after analyzing the scene in the sixth camera, this routine cycles continuously until interrupted by STOP_BLOB_TRACK or another target recognition command. BLOB_DETECT does not track; no information from previous frames is used to recognize objects in the current frame.

3.4.13 DETERMINE_GRASP

The DETERMINE_GRASP command is made to discover the regions on an object that can be grasped by the EVAR's hands or arms. Currently NO_GRASP_REGION is sent as a function, with NUL as a modifier to VISION_CTL as part of the message header.

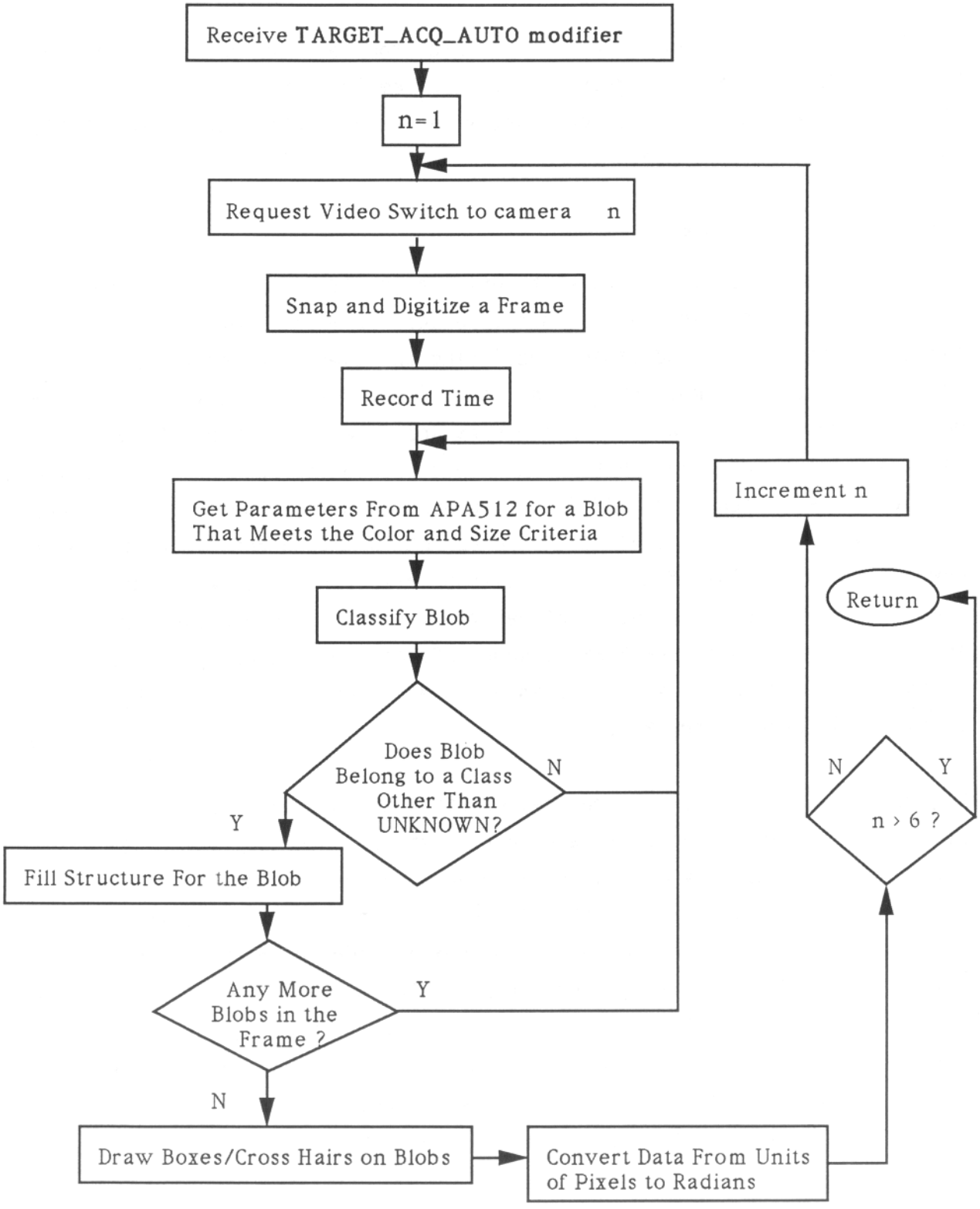

Figure 3-7.- TARGET_ACQ_AUTO functional block diagram.

3.4.14 MONITOR_GRASP

The MONITOR_GRASP command is called to track the relative positions of the EVAR's hands and arms as a grasp is being attempted. Currently NO_GRASP_REGION is sent as a function, with NUL as a modifier to VISION_CTL as part of the message header.

3.4.15 BLOB_TRACK **parm**

BLOB_TRACK is the only command that puts the APA512S in an obstacle/target tracking mode. **parm** is the sequence of data: azimuth, elevation, range, height, width, depth, and camera specifying the camera number and blob to be tracked. Usually, TARGET_ACQ_MANUAL is called beforehand to verify the target or obstacle class and obtain the **parm** specifiers. The APA512S will continuously track the object until the object's velocity exceeds a preset limit or until interrupted by VISION_CTL. The APA512S will automatically request a camera switch if it perceives that the object is moving out of the FOV of the current camera. (This feature is only supported for the three chest cameras.) The EVAR software does not yet track blobs through occlusions. The software does allow the APA512S to send information on other blobs in the FOV if they belong to a class other than UNKNOWN. However, only the object specified in **parm** is tracked.

Figure 3-8 shows a block diagram representing the procedure followed for the BLOB_TRACK command. The **parm** data must first be converted from units of radians to pixels. A video switch is requested for the camera portion of the message. Once an image is digitized, the time is recorded so that "timestamp" can be filled for each blob being reported in the frame. Each set of blob parameters is read from the APA512 one by one for either recognition or tracking. If a blob is "close enough" to the object tracked in the previous frame, the two are considered the same and tracking has continued. If the new blob is not close to the tracked blob, it is classified and reported if it belongs to a class besides UNKNOWN.

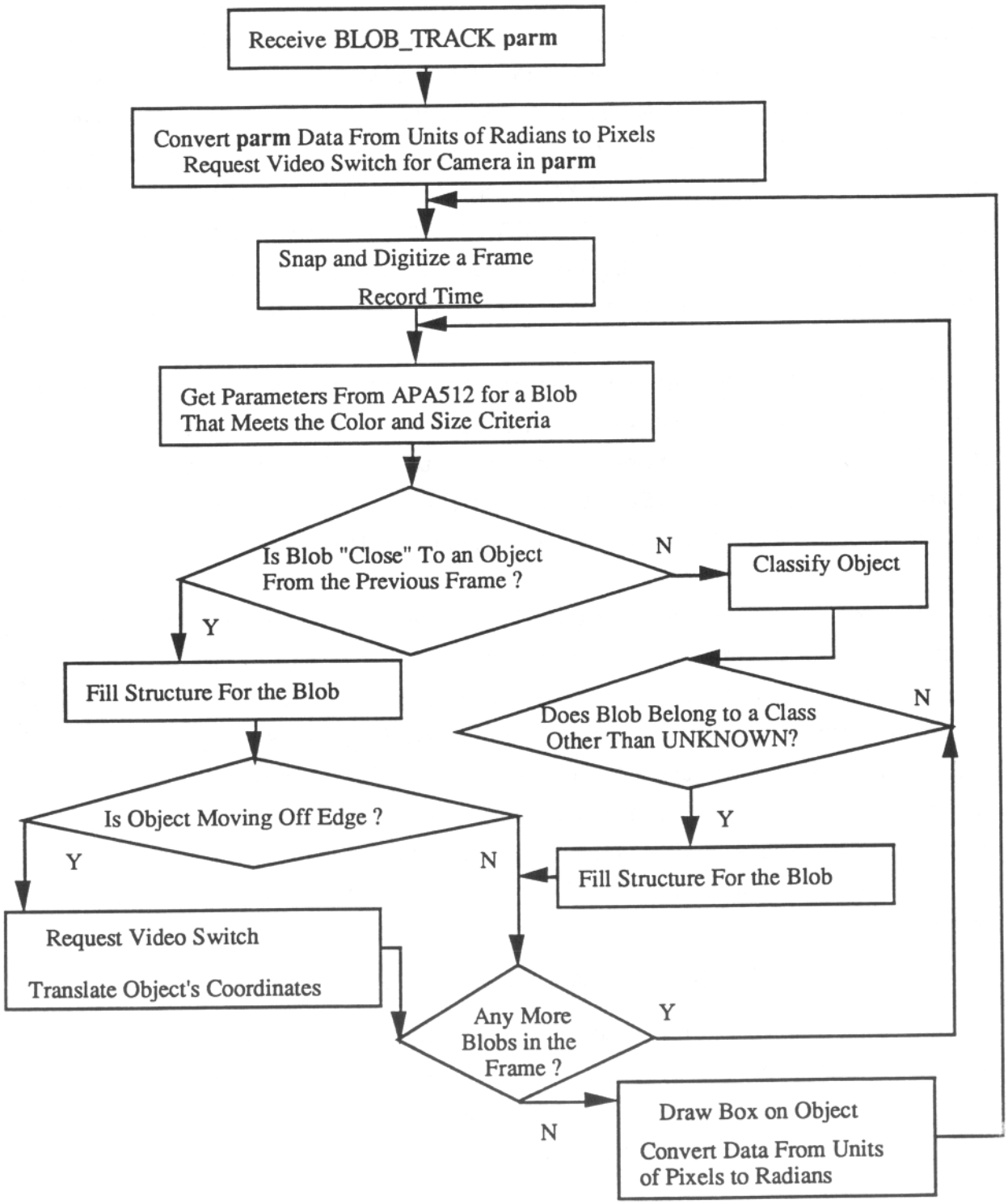

Figure 3-8.- BLOB_TRACK functional block diagram.

The key to the tracking process is to adequately define what is meant by "close." For the EVAR software, closeness is defined in a Euclidean distance sense. A blob in the current frame will be matched to an object in the previous frame if the blob is within CENT_VAR (centroid variance) pixels to both the *xcent* and *ycent* coordinates, and within DIFF_VAR (difference variance) pixels to both the *xdiff* and *ydiff* envelopes. To wisely choose these limits, one must estimate the speed and rotation of the object to be tracked. Making the limits too small may cause the APA512S to lose track because of excessive object velocities. Making the limits too large may cause the system to lock onto a residual blob that is spatially close to the object.

After the structure has been filled for the tracked object, its new centroid and envelopes are recorded so that a blob in the next frame can be matched to it. Thus, in the first frame of tracking, the new blob parameters are compared with **parm** from VISION_CTL. Henceforth, new blob parameters are compared to the last known position/envelope of the object.

A feature of the software is the ability to automatically request a camera switch and regain tracking of the object in the new camera's FOV. The three chest cameras are aligned such that there is an X_BORDER pixel overlap, as shown in figure 3-9. If the object is in CAMERA_LEFT, moving to the right, and within X_BORDER of the edge of the screen, for example, CAMERA_CENTER is requested and the object's expected location is translated the appropriate distance. In figure 3-10 a point is in the overlapping area of the cameras' FOV's. The coordinates for the left and right cameras are X_L and X_R, respectively. The equation

$$X_L + (X\_BORDER - X_R) = RES\_X$$

defines the transformation from one camera to the other, where RES_X is the pixel resolution of the camera in the x direction. If the point is moving from left to right, the equation is solved for X_R to calculate the appropriate coordinate transformation.

Finally, once all the blobs have been discarded or recorded into a structure, a bounding box is drawn around the object being tracked. If the track is

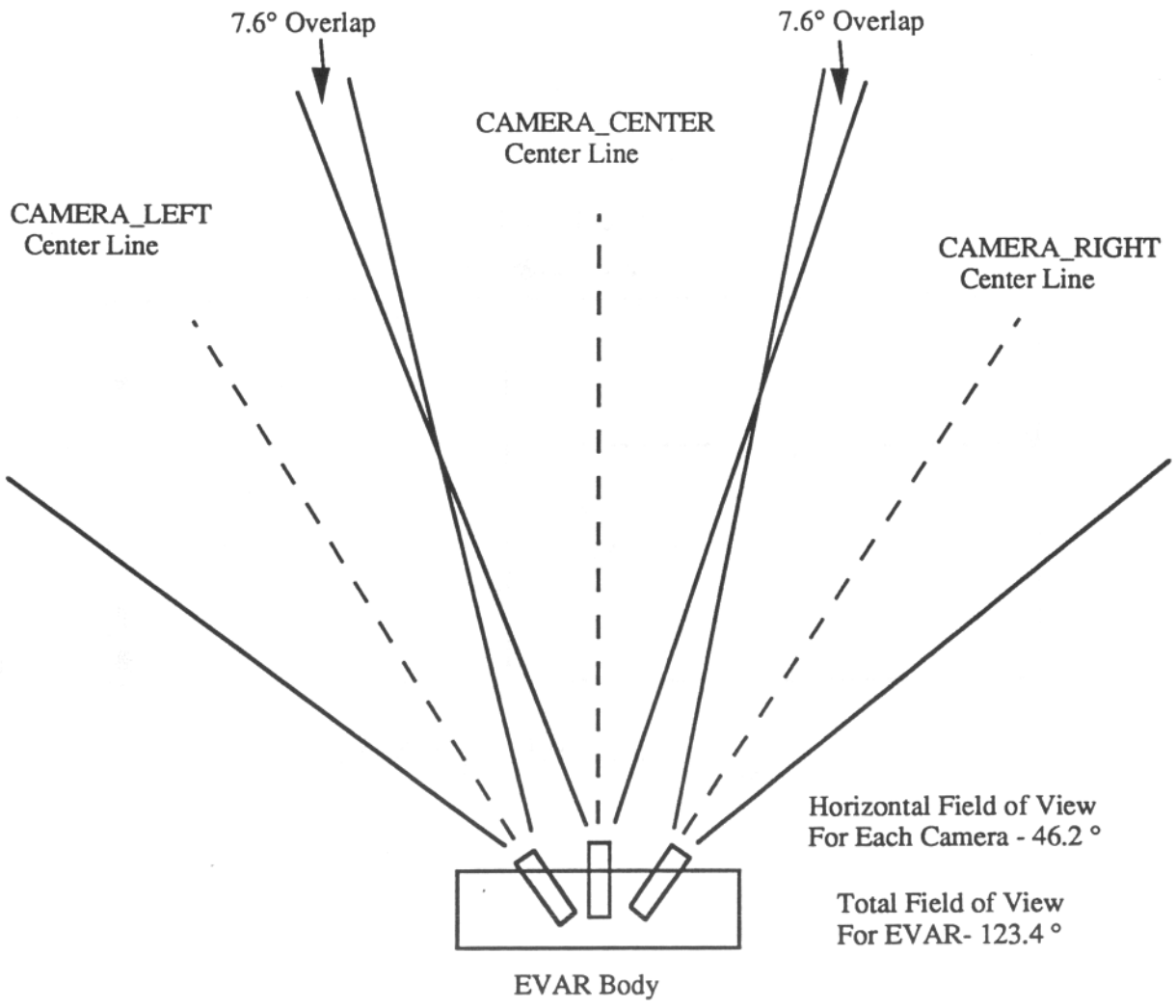

Figure 3-9.- Chest camera placement for EVAR.

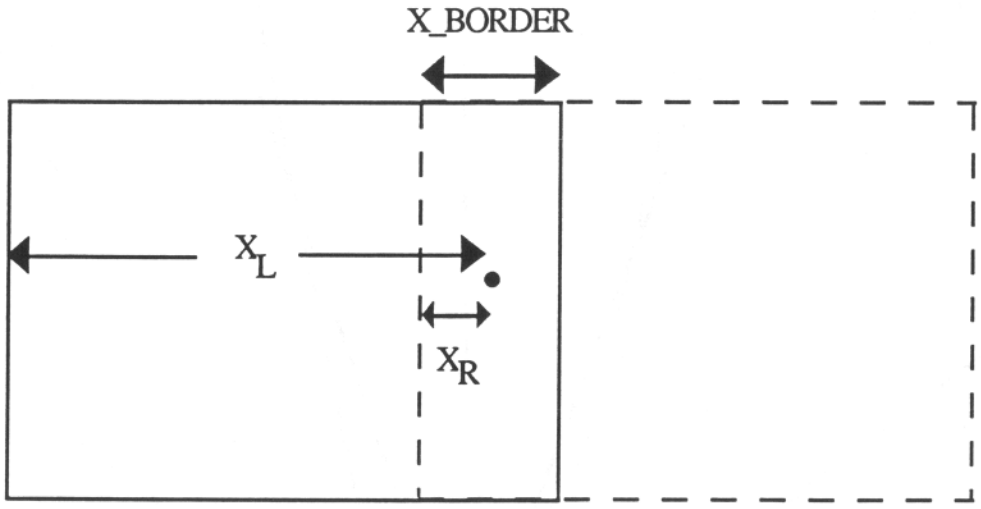

Figure 3-10.- X-coordinate transformation for a moving point.

lost, TARGET_NOT_IN_SCENE is returned. The data measurements are converted to units of radians. BLOB_TRACK can be interrupted with STOP_BLOB_TRACK or one of the three target recognition commands.

4. INPUT/OUTPUT COMMUNICATIONS INTERFACE

The APA512S machine vision processing laboratory is in the Tracking Techniques Branch (TTB) of the JSC Tracking and Communications Division (TCD) and is located in building 14. The rest of the EVAR group, along with the vision subsystem (i.e., VISION_CTL), is located in building 9A for demonstration purposes, where the robot has access to the Air Precision Bearing Floor (APBF). Communication with VISION_CTL is accomplished over an optical fiber link that connects the two buildings. The link provides incoming and outgoing paths for video, serial data at 9600 baud rate conforming to RS232C standards, and the intercom to enable the operators to talk to each other during testing and actual run of the entire system.

For any command from VISION_CTL, there is a response from the APA512S. The commands and responses that make up the protocol are sent over the communication link as a message header. The message header contains the codes to define the source, destination, and type of the function needed to be performed. The message header has the following form:

source,destination,function,modifier,length,message.

The codes are separated by commas and are all represented as integers, except for messages that can be represented as integers or real numbers. If no information is to be transmitted in the modifier code, a NUL is sent for *modifier*. *length* is the length of the message portion in bytes. All integers are 32-bit unless otherwise specified. All real numbers are 32-bit floating point. A zero message length may be sent in the case of the message header having no *message*.

Symbolic names composed of all capital letters are used to represent the source codes, destination codes, function codes, and other information passed. These names are listed in the include file *vision.h*. Each of these symbolic names is associated with some integer value that is actually used in the given message.

The source code is listed in four separate files: *evar.c*, *io_com.c*, *init.c*, and *track.c*. The main file is *evar.c*; all others are subroutines. The *io_com.c* file has software for I/O communications over the RS232 line; the *init.c* file has software to initialize, enable, and disable the image processing boards; and the *track.c* file has software for tracking algorithms. In their current configurations, the programs can read up to two message headers concatenated together and can respond to each one at a time. In order to run the program, simply type *run* at the prompt.

Figure 4-1 gives a block diagram of the I/O communications interface that links VISION_CTL with the APA512S. An initialization sequence, which is a string of characters ending with "?!", is received in order to synchronize the two systems. The APA512S then waits for a message and, once the message is received, interprets the *function* portion of the message header by calling a routine called *response()*. This function returns RESTART (level 0), PROCESS (level 1), and IGNORE or RESPOND (level 2). If the return is RESTART, the main program waits for the initialization sequence once again. If the return is IGNORE, the message is ignored altogether because it does not make any sense to the APA512S; the main program waits for another message. If the return is RESPOND, the program sends back the requested information (e.g., health status) or sets the APA512S parameters (e.g., threshold level and time stamp). If PROCESS is returned, one of the four basic level 1 routines (TARGET_ACQ_MANUAL, TARGET_ACQ_AUTO, BLOB_DETECT, or BLOB_TRACK) is executed. The basic operation of each of these routines has been explained in detail in section 3. First a video switch is requested by sending B14_VIDEO_SWITCH_REQUEST to VISION_CTL. The message received from VISION_CTL is passed through the DECISION BOX (figure 4-2). In the DECISION BOX (used for TARGET_ACQ_MANUAL, TARGET_ACQ_AUTO, and BLOB_DETECT only) RESTART (level 0), PROCESS (level 1), IGNORE (level 2), RESPOND (level 2), and/or VIDEO_SWITCHED are returned, depending upon the *function* code received from the VISION_CTL. It allows one to either get out of an infinite tracking loop or stay in one, depending on what is received.

The appropriate action is taken to responses RESTART, PROCESS, and IGNORE or RESPOND, as before. If VIDEO_SWITCHED is the response, usually after a

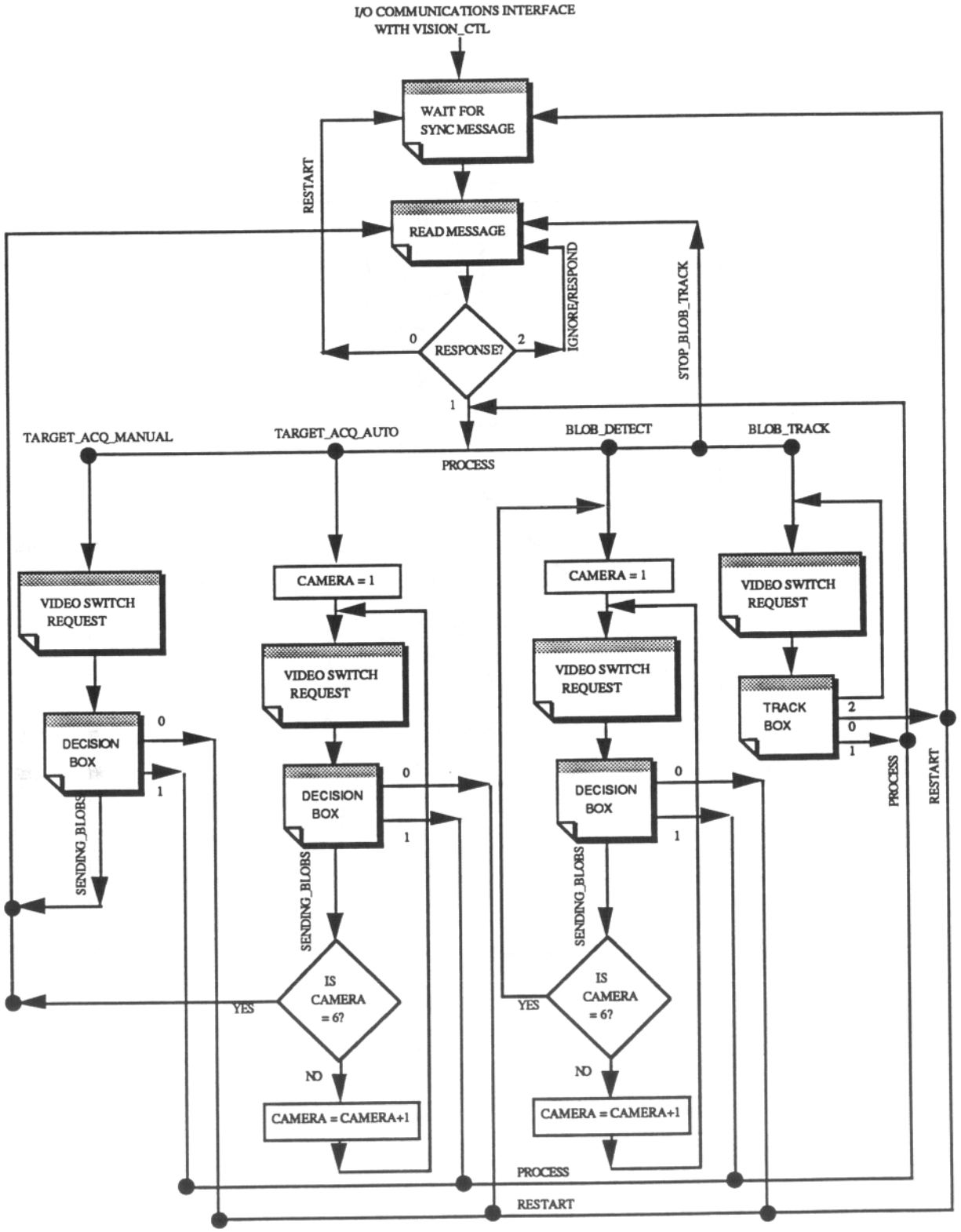

Figure 4-1- Block diagram depicting I/O communications interface of APA512S with VISION_CTL.

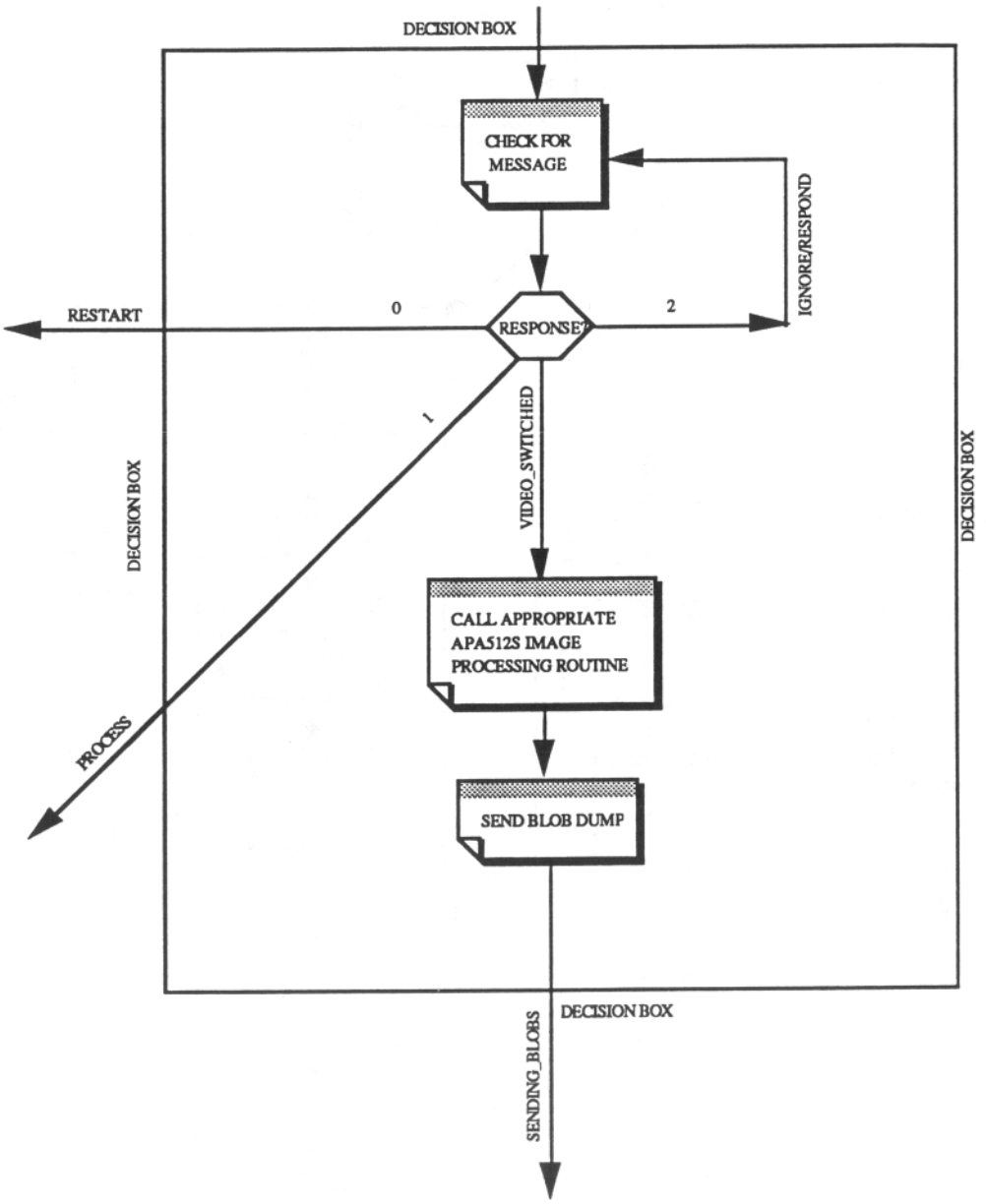

Figure 4-2.- Block diagram of DECISION BOX.

video_switch() operation, then the program starts processing images in the new (or switched) camera's FOV.

When PROCESS is returned (usually a STOP_BLOB_TRACK from VISION_CTL to stop the basic tracking routines), the current process is exited and the new process (level 1) is executed immediately.

In the case of BLOB_TRACK, after a video switch request, control is passed to the TRACK BOX, from which RESTART, PROCESS, or level 2 (like video switching requested) are returned (figure 4-3). In the TRACK BOX a message is read after a video switch request is made to VISION_CTL. The response to the message is RESTART, PROCESS, IGNORE or RESPOND, or VIDEO_SWITCHED. For the first three responses the appropriate action is taken as before. If the response is VIDEO_SWITCHED, the appropriate blob-tracking routine is called and the blob structure is sent to VISION_CTL. Once again, there is a message check. If there is a valid message, it is checked. The response to this message is once again RESTART, PROCESS, or IGNORE or RESPOND, and the appropriate action is taken as before. If there is no message waiting to be read, the main program determines if a video switch is required in case the tracked blob goes out of view of the current camera. If not, the process of blob tracking is continued as before. If a video switch is required, then RESPOND is returned, the current process is suspended temporarily, a video switch is requested, and the process is initiated again.

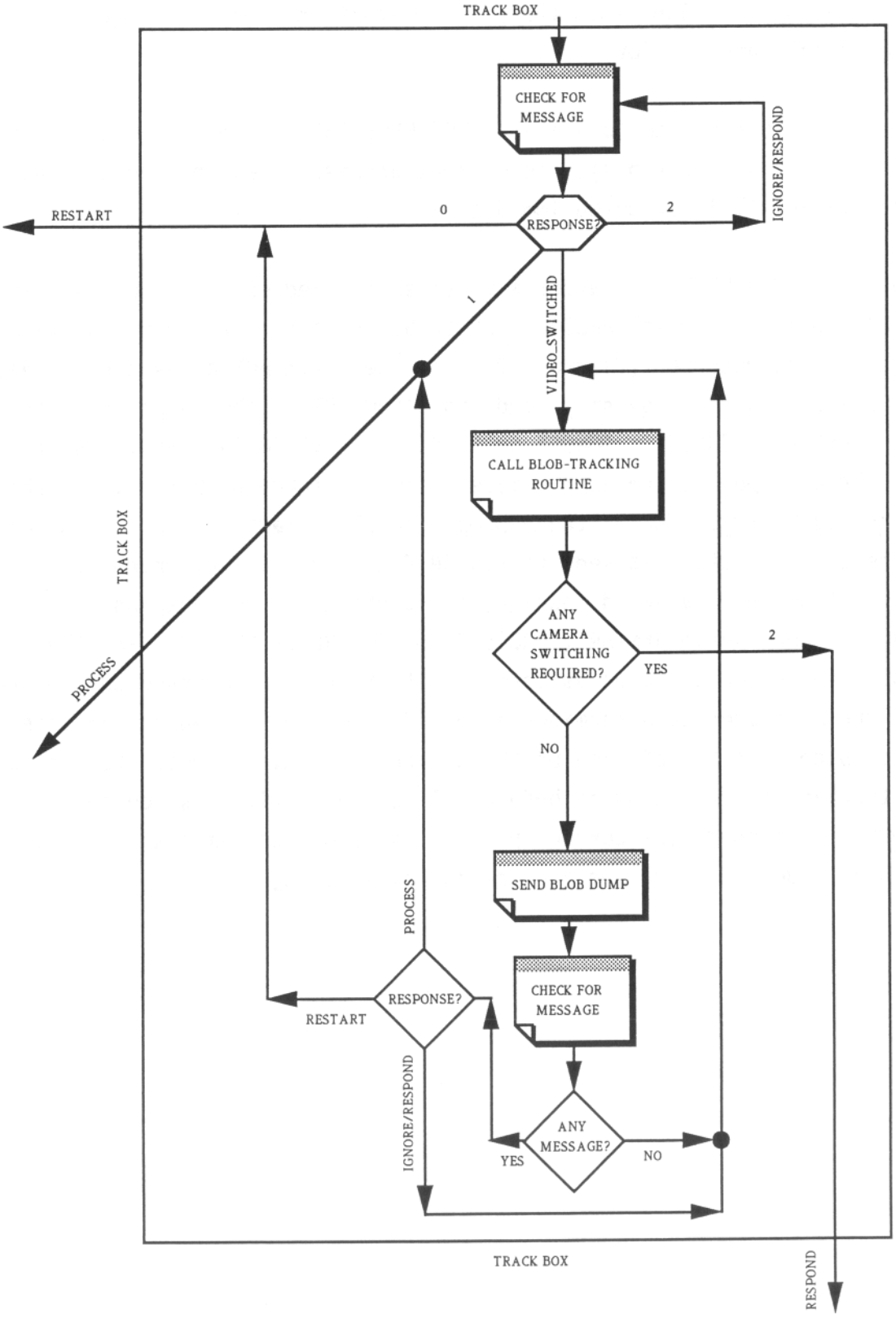

Figure 4-3.- Block diagram of TRACK BOX.

5. CONCLUSIONS

The goal of the target recognition, target tracking, and communications I/O interface software was to develop an intelligent image processing and tracking system capable of providing vision support for the EVAR and other systems. Video from one of six cameras is received from the EVAR in JSC building 9A via a fiber optic link. The interfacing and communications software enables the APA512S system to receive vision processing commands and send data specifying the positions, sizes, and classifications of objects. The APA512S can also track a specified object. Two-dimensional target recognition is accomplished with an artificial neural network.

The APA512S, consisting of four Datacube, Inc., boards for preprocessing and an APA512 board set for blob analysis, returns descriptive parameters about each blob in the EVAR's FOV. Some of these parameters are used for object recognition, and others, such as *area* and *centroid*, are sent to VISION_CTL.

The APA512S parameters used for object recognition are *axratio* and *peround*. These two features are inputs to the neural network that classifies the object into ASTRONAUT, WRENCH, ORU, and NEITHER categories.

This system can track an object as it moves from the FOV of one camera to the FOV of another.

The purpose of the I/O interface and communications software is to interpret the message from VISION_CTL in JSC building 9A, call the appropriate image processing subroutine, and send the recognition/tracking information back to VISION_CTL.

This vision tracking system provides performance superior to the McDAC system. The McDAC Tracker does not perform object recognition, whereas the APA512S recognizes four classes of objects: astronauts, wrenches, ORU's, and unknown obstacles. Also, the McDAC Tracker is limited to processing information on 12 blobs at most. The APA512 can return information on up to 256 blobs.

In future work the vision tracking system should be extended to recognize three-dimensional objects, with tracking of the objects through occlusions under typical space lighting conditions. When the time comes to place the hardware directly into the VME bus card cage of the EVAR, the interfacing software must be revised since data will no longer be sent via the RS232 line. The only hardware change necessary is to set the jumper pin connections on the boards to the appropriate EVAR system configuration.

6. REFERENCES

1. Extravehicular Activity (EVA) Retriever Program Plan, Basic. NASA/JSC (Houston, TX), JSC-22144, May 1987.

2. Extravehicular Activity (EVA) Retriever Software Interface Requirements, Phase II. Lockheed Engineering & Sciences Company (Houston, TX), contract NAS 9-17900, job order 25-140, LESC-26075, January 1989.

3. DIGIMAX User's Manual. Datacube, Inc. (Peabody, MA), UM0041-6.1, 1988.

4. VFIR User's Manual. Datacube, Inc. (Peabody, MA), UM0043-4.1, 1988.

5. MAX-SP User's Manual. Datacube, Inc. (Peabody, MA), UM0046-5.0, 1988.

6. MAXGRAPH User's Manual. Datacube, Inc. (Peabody, MA), UM0055-3.0, 1988.

7. APA512-MX Area Parameter Accelerator User Manual. Issue 2.2 (Adelaide, S.A., Australia), 1987.

8. Ballard, D. H.; and Brown, C. M.: Computer Vision. Prentice Hall, Inc. (Englewood Cliffs, NJ), 1982.

9. Pao, Y.: Adaptive Pattern Recognition and Neural Networks. Addison-Wesley Publishing Co., Inc. (Reading, MA), 1989.

10. Duda, R.; and Hart, P.: Pattern Classification and Scene Analysis. John Wiley and Sons (New York, NY), 1973.

11. Khanna, T.: Foundations of Neural Networks, Addison-Wesley Publishing Co., Inc. (Reading, MA), 1990.

APPENDIX A

SOFTWARE DOCUMENTATION

APPENDIX A

SOFTWARE DOCUMENTATION

BLOB_TRACK

DESCRIPTION

This function is called by *main()* in *evar.c* when the command received in the *function* code of the message header from VISION_CTL is BLOB_TRACK **parm.** Its responsibility is to track the target specified in **parm,** and it has the ability to request camera switching for any of the three chest cameras.

SYNTAX

int blob_track(no_blobs, camera)
*int *no_blobs, *camera;*

RETURN VALUES

TARGET_NOT_IN_SCENE - The object specified by the parameters in **parm** could not be found.

SUCCESS - The object was tracked.

NO_BLOBS_FOUND - No blobs are in the FOV.

CHECK_MESSAGE

DESCRIPTION

This function checks, but does not wait, to see if there is any valid message waiting to be read, and then reads the whole message into a buffer. It returns all the values explained in the *response()* operation. If there is no message waiting to be read, it returns NO_MESSAGE.

SYNTAX

int check_message()

CLASSIFY

DESCRIPTION

This function uses a neural network to classify blobs into one of three classes (ASTRONAUT, WRENCH, or ORU) based on the globally defined parameters *axratio* and *peround*. It is called by *image_process()* and *blob_track()* in *track.c*. *classify()* can be found in *track.c*.

SYNTAX

classify()

RETURN VALUES

None

DECISION

DESCRIPTION

This function first counts the number of commas to determine how many message headers were received. The program currently accepts up to two message headers concatenated together and responds to these one at a time. This function makes a call to *response()*.

SYNTAX

int decision()

RETURN VALUE

READ_AGAIN - The message read is not complete (i.e., the number of commas in the message header is less than five).

DISABLE_BOARDS

DESCRIPTION

This function is invoked when the command received in the *function* code of the message header from VISION_CTL is DISABLE or RESET. The program is ready to receive the sync initialization message after this function is executed.

SYNTAX

disable_boards()

RETURN VALUES

FAILURE - The APA512S boards were not disabled correctly.
SUCCESS - The boards were disabled.

DRAW_BOX

DESCRIPTION

This function uses the Datacube MAXGRAPH board to draw recognition symbology on the specified globally defined blob number. Boxes are drawn around objects belonging to the ASTRONAUT class, cross hairs are drawn on WRENCH objects, and ORU objects have both boxes and cross hairs. *draw_box* is listed in *track.c*.

SYNTAX

draw_box()

RETURN VALUES

None

ENABLE_BOARDS

DESCRIPTION

This function is invoked when the command received in the *function* code of the message header from VISION_CTL is ENABLE. *enable_boards* is in *init.c*.

SYNTAX

enable_boards()

RETURN VALUES

FAILURE - The APA512S boards were not enabled.
SUCCESS - The boards were enabled.

ERASE_BOX

DESCRIPTION

Drawing with the MAXGRAPH board is an exclusive OR (XOR) kind of operation; twice drawing a box erases it. This function erases symbology drawn on objects in the previous frame so that new symbology can be drawn in the current frame. *erase_box* is listed in *track.c*.

SYNTAX

erase_box()

RETURN VALUES

None

HELP_MESSAGE

DESCRIPTION

This function gives certain command line options to run the program.

SYNTAX

help_message()

IMAGE_PROCESS

DESCRIPTION

This function is invoked when the command received in the *function* code of the message header from VISION_CTL is TARGET_ACQ_MANUAL, TARGET_ACQ_AUTO, or BLOB_DETECT. Its responsibility is to give information on targets and obstacles in the FOV of the camera selected. The amount of information depends on the *function* code given. *no_blobs* is the number of obstacles/ targets that is being reported to VISION_CTL. *image_process()* is called by *main()* in *evar.c* and is listed in *track.c*.

SYNTAX

int image_process(no_blobs)
*int *no_blobs;*

RETURN VALUES

NO_BLOBS_FOUND - No blobs are in the FOV.

TARGET_NOT_IN_SCENE - The target specified by the VISION_CTL command TARGET_ACQ_AUTO or BLOB_DETECT could not be found.

SUCCESS - All tasks were successfully completed.

INITIALIZE_BOARDS

DESCRIPTION

This function reads the hardware configuration description, initializes the data base with specific hardware information, and initializes the hardware. *initialize_boards()* is called by *main()* in *evar.c* and is listed in *init.c*.

SYNTAX

int initialize_boards()

RETURN VALUES

None

INITIALIZE_IO_READ

DESCRIPTION

This function will return FAILURE if the I/O port is not opened successfully for a *read* operation; otherwise, it returns a path number used to identify the file when I/O is performed.

SYNTAX

int initialize_io_read()

INITIALIZE_IO_WRITE

DESCRIPTION

This function will return FAILURE if the I/O port is not opened successfully for a *write* operation; otherwise, it returns a path number used to identify the file when I/O is performed.

SYNTAX

int initialize_io_write()

PIXEL_TO_RADIAN

DESCRIPTION

This function changes (x,y) pixel blob centroid coordinates of *no_blobs* blobs into radians with the following transformation for the CCD Elmo camera:

azm_centroid = FOV_X * (π/180)/RES_X *(*azm_centroid* - RES_X/2)
elv_centroid = FOV_Y * (π/180)/RES_Y *(*elv_centroid* - RES_Y/2)
height_env = FOV_Y * (π/180)/RES_Y * *height_env*
width_env = FOV_Y * (π/180)/RES_Y * *width_env*
azm_env_center = FOV_Y * (π/180)/RES_Y*(*azm_env_center* - RES_X/2)
elv_env_center = FOV_Y * (π/180)/RES_Y * (*elv_env_center* - RES_Y/2)
azm_velocity = FOV_Y * (π/180)/RES_Y * *azm_velocity*
elv_velocity = FOV_Y * (π/180)/RES_Y * *elv_velocity*

FOV_X - Elmo camera FOV in x direction in degrees
FOV_Y - Elmo camera FOV in y direction in degrees
RES_X - Digitizer pixel resolution in x direction
RES_Y - Digitizer pixel resolution in y direction

The APA512S origin, located in the upper left corner, must be translated to the center first. Then the pixel units must be transformed into radian units. This function is listed in *track.c*.

SYNTAX

int pixel_to_radian(no_blobs)
int no_blobs;

RETURN VALUES

None

PRINT_MESSAGE

DESCRIPTION

This function prints out the message header read from the I/O port.

SYNTAX

int print_message()

RADIAN_TO_PIXEL

DESCRIPTION

This function changes radian units into (x,y) pixel units, just opposite to the *pixel_to_radian()* operation, when BLOB_TRACK **parm** is received from VISION_CTL. The following transformations are calculated:

azm_centroid = *azm_centroid*/(FOV_X * (π/180)/RES_X) + RES_X/2
elv_centroid = *azm_centroid*/(FOV_Y * (π/180)/RES_Y) + RES_Y/2
height_env = *height_env*/(FOV_Y * (π/180)/RES_Y)
width_env = *width_env*/(FOV_X * (π/180)/RES_X)

SYNTAX

int *radian_to_pixel()*

RETURN VALUES

None

READ_INITIALIZATION

DESCRIPTION

This function returns SUCCESS if a sync message ending with "?!" is received; otherwise, it returns FAILURE.

SYNTAX

int read_initialization()

DESCRIPTION

This function performs an I/O *read* operation, and the message read is stored in a buffer.

SYNTAX

int read_message()

DESCRIPTION

This function returns IGNORE, RESPOND, RESTART, PROCESS, and VIDEO_SWITCHED, depending upon the *function* code received from VISION_CTL. The various possible commands and their responses are as shown in the following table:

Command (function code)	Response
0	IGNORE
PCIF_START_SENDING_VISUAL_RESULTS	IGNORE
PCIF_STOP_SENDING_VISUAL_RESULTS	IGNORE
SET_THRESHOLD_VALUE	RESPOND
ENABLE	RESPOND
DISABLE	RESTART
RESET	RESTART
ARE_YOU_HERE_TODAY	RESPOND
SEND_HEALTH_STATUS	RESPOND
IDLE	IGNORE
RESUME	IGNORE
START_PROCESSING	IGNORE
SINGLE_FRAME	IGNORE
SET_TIME	RESPOND
B14_VIDEO_SELECT_COMPLETE	VIDEO_SWITCHED
STOP_BLOB_TRACK	PROCESS
BLOB_DETECT_AND_TRACK	PROCESS (no action)
TARGET_ACQ_MANUAL	PROCESS
TARGET_ACQ_AUTO	PROCESS
BLOB_DETECT	PROCESS
BLOB_TRACK	PROCESS
DETERMINE_GRASP	RESPOND
MONITOR_GRASP	RESPOND

SYNTAX

int response()

SECOND_MESSAGE

DESCRIPTION

This function responds to the second part of the message header if the two messages read are concatenated.

SYNTAX

int second_message()

SEND_BLOB_DUMP

DESCRIPTION

This function puts the blob dump for *val* blobs in the *message* portion of the message header and then sends it to VISION_CTL with SENDING_BLOBS (long structure for blobs) in the *function* code of the message header.

SYNTAX

```
int   send_blob_dump(val)
int   val;
```

SEND_MESSAGE

DESCRIPTION

This function sends a variety of messages to VISION_CTL, based on return *ret* from the tracking algorithms and *no_blobs* blobs.

SYNTAX

int send_message(ret, no_blobs)
int ret, no_blobs;

SEND_RETURN

DESCRIPTION

This function writes ordinary messages to the opened I/O port where there is no *message* portion of the message header. *ret* and *mod* represent the function and the modifier codes in the message header, respectively.

SYNTAX

int send_return(ret, mod)
int ret, mod;

VIDEO_SWITCH

DESCRIPTION

This function requests VISION_CTL for a video switch by sending out B14_VIDEO_SELECT_CHANNEL as the *function* code in the message header. It returns all the values explained in the *response()* operation.

SYNTAX

int *video_switch()*

APPENDIX B

SOURCE CODE

```
/*********************************************************************
*  Evar.c                                                           *
*                                                                   *
*  Authors  :    Sunil Fotedar & Alan Smith                         *
*                LESC, Houston, TX                                  *
*********************************************************************/

/* Header files */

#include <stdio.h>
#include <time.h>
#include <modes.h>
#include <std_def.h>
#include <vfConfig.h>
#include <kernel.h>
#include <apHead.h>
#include <blobHead.h>
#include <discrim.h>
#include <aai.h>
#include <user.h>
#include <vision.h>

int ctrl_C=FALSE;

char     message_buffer[WIDE], tempmsg[WIDE];

struct   blob_structure bst[BLOBS];
struct   blob_structure_short sst[BLOBS];
Message  msg;

PARAM_FLAGS     flags;
BLOB_DESC       *bd1, *bd2, *bdtemp;
CONNECT_BIDS    boards;
int             conFlags;

long     from   = (long)(APA512S);    /* source identification */
long     to     = (long)(VISION_CTL);  /* destination identification */
long       dummy1=0, dummy2=0, dummy3=0, dummy4=0, dummy5=0;
int     display_data = NO;
int     time_diff = 0;
int      threshold = 0x80;
int     floor_line = 0x1e0;
int      target_acq_auto_flag = UNSET, frame_count;
int      ret_en=-1, comma_count;
```

```
int       read_port, write_port;
/* Beginning of main() */

main(argc,  argv)

int     argc;
char    **argv;

{
    int          i, j, count, model, camera, ret, ret1=-99, no_blobs, code;
    int          cam=0, time_ret, respond_ret, blob_ret;
   long         cmd, val, len, funct;

  /* Command-line options */

if(argc > 1) {
       if((strncmp(argv[1],"HELP",4) == SUCCESS) ||
             (strncmp(argv[1],"help",4) == SUCCESS) ||
            (strncmp(argv[1],"?",1) == SUCCESS)) {
                help_message();
               exit(0);
      }
      for(i=1;i<argc;++i) {
           if (strncmp(argv[i],"DATA",4) == SUCCESS) display_data = YES;
            if (strncmp(argv[i],"data",4) == SUCCESS) display_data = YES;
      }
  }

/* Open I/O port */

        if((read_port=initialize_io_read())==FAILURE) {        /* open I/O port to
read  */
          printf("APA512S Problem in opening the I/O port.\n");
        goto apa512s_exit;
  }

    if((write_port=initialize_io_write())==FAILURE) {        /* open I/O port to
write  */
          printf("APA512S Problem in opening the I/O port.\n");
        goto apa512s_exit;
  }

/* Initilialize APA512S boards */
```

```
    if((ret=initialize_boards())!=SUCCESS) {
         printf("APA512S Problem in initializing APA512S boards.\n");
        goto apa512s_exit;
   }

/* Initialize buffers */

          memset(message_buffer,0x00,WIDE);
         memset(tempmsg,0x00,WIDE);
        memset(bst,0x00,BLOBS);
        memset(sst,0x00,BLOBS);

/* Wait for initialization message */

    initialize:

    target_acq_auto_flag = UNSET;

     printf("\nAPA512S Waiting for RS232 initialization sequence!!!\n");

       if((ret=read_initialization())!=SUCCESS) {
            printf("APA512S Problem in reading the RS232 initialization
sequence.\n");
          goto apa512s_exit;
     }
         printf("\nAPA512S Message received: %s\n",message_buffer);
          if((ret=write(write_port,message_buffer,strlen(message_buffer)))<0) {
            printf("APA512S Problem in sending data out.\n");
          goto apa512s_exit;
     }
       send_return(I_AM_HERE_TODAY,NUL);

           printf("\nAPA512S Ready to go to work!!!\n");

while(1)

{
     memset(message_buffer,0x00,WIDE);
    memset(tempmsg,0x00,WIDE);
   memset(bst,0x00,BLOBS);

    printf("\nAPA512S Waiting for a command from Vision Subsystem!!!\n");

    respond_ret = read_message();
```

```
/* we don't want IGNORE in this file */
        if(respond_ret==RESTART) goto initialize;

/* Handle all other commands */

    commands:

    target_acq_auto_flag = UNSET;

switch((int)(msg.functn))  {

     case   STOP_BLOB_TRACK :

               printf("APA512S Blob Track is stopped.\n");
              second_message();
             break;

     case   BLOB_DETECT_AND_TRACK :

               printf("APA512S BLOB_DETECT_AND_TRACK is received. No
action  is  taken.\n");
                second_message();
            break;

     case   TARGET_ACQ_MANUAL :
                 /* msg.modifer = camera # = sourceid */

              (int)bst[0].sourceid = (int)msg.modifer;
            if(((int)bst[0].sourceid<CAMERA_LEFT)||
              ((int)bst[0].sourceid>CAMERA_MAPPER_VIDEO))
                   (int)bst[0].sourceid = CAMERA_CENTER;

               ret = video_switch();
             if(ret==VIDEO_SWITCHED) {
                    printf("\nAPA512S Video is from Camera# %d.\n",
                          (int)bst[0].sourceid);

                  ret = image_process(&no_blobs);

                   send_message(ret,no_blobs);

          }
            else if(ret==RESTART) goto initialize;
            else if(ret==PROCESS) goto commands;
```

```
          second_message();
          break;

case    TARGET_ACQ_AUTO :

        target_acq_auto_flag = SET;
        model = (int)msg.modifer;    /* model */

    for(cam=CAMERA_LEFT;cam<=CAMERA_LHAND;cam++)
    {
          (int)bst[0].sourceid = cam;
          ret = video_switch();

        if(ret==VIDEO_SWITCHED) {
                  printf("\nAPA512S Video is from Camera# %d.\n",cam);

                bst[0].sourceid = cam;
                bst[0].idcode = model;
                 ret = image_process(&no_blobs);

                  send_message(ret,no_blobs);

        }
          else if(ret==RESTART) goto initialize;
          else if(ret==PROCESS) goto commands;

             if((ret1=check_message())!=NO_MESSAGE) {
                if(ret1==RESTART) goto initialize;
                 else if(ret1==PROCESS) goto commands;
          }
    }
      target_acq_auto_flag = UNSET;
      second_message();
     break;

case    BLOB_DETECT :

       for(cam=CAMERA_LEFT;cam<(CAMERA_LHAND+2);cam++) {

          if(cam==(CAMERA_LHAND+1)) cam=CAMERA_LEFT;

                (int)bst[0].sourceid = cam;

               ret = video_switch();
```

```
            if(ret==VIDEO_SWITCHED) {
                        printf("\nAPA512S Video is from Camera# %d.\n",cam);

                    (int)bst[0].sourceid = cam;

                     ret = image_process(&no_blobs);

                     send_message(ret,no_blobs);

            }
              else if(ret==RESTART) goto initialize;
              else if(ret==PROCESS) goto commands;
                 if((ret1=check_message()) != NO_MESSAGE) {
                    if(ret1==RESTART) goto initialize;
                     else if(ret1==PROCESS) goto commands;
                }
                  second_message();

            }    /* end of infinite loop */

              break;

      case   BLOB_TRACK :

                memset(tempmsg,0x00,WIDE);
             sprintf(tempmsg,"%ld,%ld,%ld,%ld,%ld,",
                        msg.source,msg.destin,msg.functn,msg.modifer,msg.length);

                sscanf(message_buffer+strlen(tempmsg),"%f,%f,%f,%f,%f,%f,%ld,",
                     &bst[0].azm_centroid,&bst[0].elv_centroid,
                     &bst[0].rng_centroid,&bst[0].height_env,
                      &bst[0].width_env,&bst[0].depth_env,&bst[0].sourceid);

              printf("APA512S Parameters for blob-tracking are:
\n%f,%f,%f,%f,%f,%f,%ld,\n",
                     bst[0].azm_centroid,bst[0].elv_centroid,
                     bst[0].rng_centroid,bst[0].height_env,
                      bst[0].width_env,bst[0].depth_env,bst[0].sourceid);

              if(((int)bst[0].sourceid<CAMERA_LEFT)
                 ||((int)bst[0].sourceid>CAMERA_MAPPER_VIDEO))
                       (int)bst[0].sourceid = CAMERA_CENTER;
```

```
            radian_to_pixel();

            ret = video_switch();

            if(ret==VIDEO_SWITCHED) {  /* 0x1116 received */
                printf("\nAPA512S Video is from Camera#
%d.\n",(int)bst[0].sourceid);

              frame_count = 0;
                          /* Reset frame counter */
          for(;;) {

            track:

            cam = (int)bst[0].sourceid;

          blob_ret = blob_track(&no_blobs,&camera);

            (int)bst[0].sourceid = camera;

             if(camera != cam) {
                ret = video_switch();
              if(ret==VIDEO_SWITCHED) {
                      printf("\nAPA512S Video switched to Camera# %d.\n",
                            (int)bst[0].sourceid);
                  goto track;
              }
                else if(ret==RESTART) goto initialize;
                else if(ret==PROCESS) goto commands;
              } /* end of if(camera) */

               send_message(blob_ret,no_blobs);

          if((ret1=check_message())!=NO_MESSAGE) { /* to get out of infinite
loop */
            if(ret1==RESTART) goto initialize;
            else if(ret1==PROCESS) goto commands;
            else goto track;
        }
             } /* end of infinite loop */
           } /* end of if(ret==VIDEO_SWITCHED) after initial video switch */
            else if(ret==RESTART) goto initialize;
            else if(ret==PROCESS) goto commands;
```

```
            break;

          default:
            if(respond_ret==NOT_UNDERSTOOD)
                    printf("\nAPA512S Command not understood. Try Again!!!\n");
                second_message();
            break;
    }

}  /* End of while(1) */

apa512s_exit:                   /* exit the program */

      printf("\nAPA512S Exiting ... Try again!!!\n\n");
      free(tempmsg);
       free(message_buffer);
     free(bst);
     free(sst);
      close(read_port); /* close the read port */
      close(write_port); /* close the write port */
    exit(0);

} /* End of main() */

/*************************************************************************/
```

```
/*********************************************************************
*  Io_com.c                                                          *
*                                                                    *
*  Author  :      Sunil Fotedar                                      *
*                 LESC, Houston, TX                                  *
*********************************************************************/

/* Header files */

#include <stdio.h>
#include <time.h>
#include <modes.h>
#include <vision.h>

extern char message_buffer[], tempmsg[];
extern long from, to;
extern int time_diff, display_data, comma_count, threshold, floor_line, ret_en,
read_port,  write_port;
extern struct blob_structure bst[];
extern struct blob_structure_short sst[];
extern Message msg;

extern long dummy1,dummy2,dummy3,dummy4,dummy5;
int     char_count;
                /* No. of characters waiting in the receive queue to be read */

/*********************************************************************/
/* To open an I/O Port                                               */
/*********************************************************************/

initialize_io_read()

{
    if((read_port=open(IO_COM,S_IREAD)) < 0) {
        return(FAILURE);
   }
      return(read_port);
}

/*********************************************************************/

initialize_io_write()
{
    if((write_port=open(IO_COM,S_IWRITE)) < 0) {
```

```
        return(FAILURE);
    }
    return(write_port);
}

/*************************************************************************/
/* Read in the initialization message, and verify.                       */
/*************************************************************************/

read_initialization()
{
    int     pos;

      memset(tempmsg,0x00, WIDE);

    for(;;)
    {
        if((char_count=_gs_rdy(read_port))>0)    {
            memset(message_buffer,0x00,WIDE);
             read(read_port,message_buffer,char_count);
            strcat(tempmsg,message_buffer);
           for(pos=0;pos<strlen(tempmsg);pos++) {
               if(((tempmsg[pos]=='!')&&(tempmsg[pos-1]=='?'))
                   ||((tempmsg[pos]=='?')&&(tempmsg[pos-1]=='!')))
               goto get_out;
        }
      }  /* end of if */
    }   /* end of infinite loop */

  get_out:

    memset(message_buffer,0x00,WIDE);
     strcpy(message_buffer,tempmsg);
  return(SUCCESS);

}

/*************************************************************************/
/* Read an incoming message completely.                                  */
/*************************************************************************/

read_message()

{
```

```
    int     pos, ret;

          memset(tempmsg,0x00, WIDE);

    for(;;)
      {
            if((char_count=_gs_rdy(read_port))>0)    {
                memset(message_buffer,0x00,WIDE);
                  read(read_port,message_buffer,char_count);
                 strcat(tempmsg,message_buffer);
              comma_count=0;
                for(pos=0;pos<strlen(tempmsg);pos++) {
                   if(tempmsg[pos] == ',') comma_count++;
            }
                if((ret=decision())!=READ_AGAIN) return(ret);

          }  /* end of if */
        }  /* end of infinite loop */
}

/*********************************************************************/
/* Send Returned Values.                                             */
/*********************************************************************/

send_return(retu,   mod)

int     retu, mod;


{
   int     ret;

      memset(tempmsg,0x00,WIDE);

       memset(message_buffer,0x00,WIDE);

sprintf(message_buffer,"%ld,%ld,%ld,%ld,%ld,%s",from,to,(long)retu,(long)mod,
0l,tempmsg);
        if((ret=write(write_port,message_buffer,strlen(message_buffer)))<0)
          printf("APA512S There's a problem in sending data out.\n");

}
/*********************************************************************/
/* Send blob dump.                                                   */
```

```
/*****************************************************************/

send_blob_dump(val)

int     val;

{
    int     i, j, k, delta, ret, cam, send_message();
   long     cmd, len, funct;

  cmd  = (long)(SENDING_BLOBS);

     pixel_to_radian(val);

      memset(message_buffer,0x00,WIDE);
     memset(tempmsg,0x00,WIDE);

    switch((int)(cmd))  {

      case 0l :

            printf("APA512S Exiting... Try Again.\n");
          exit(0);

     case SENDING_BLOBS :

          memset(tempmsg,0x00,WIDE);

     if(val>MAXBLOBS) val = MAXBLOBS;

for(i=0;i<val;i++) {
      sprintf(&tempmsg[strlen(tempmsg)],

"%ld,%ld,%ld,%ld,%ld,%ld,%ld,%ld,%f,%f,%.1f,%.1f,%.1f,%.1f,%f,%f,%.1f,%f,%f,%.1f,%f
,%f,%.1f,%.1f,%.1f,%.1f,",
          bst[i].blobnum,bst[i].lblobnum,bst[i].procid,bst[i].timestamp,
         bst[i].sourceid,bst[i].idclass,bst[i].idcode,bst[i].quality,
          bst[i].azm_centroid,bst[i].elv_centroid,bst[i].rng_centroid,
          bst[i].roll_centroid,bst[i].pitch_centroid,bst[i].yaw_centroid,
          bst[i].height_env,bst[i].width_env,bst[i].depth_env,
           bst[i].azm_env_center,bst[i].elv_env_center,bst[i].rng_env_center,
         bst[i].azm_velocity,bst[i].elv_velocity,bst[i].rng_velocity,
         bst[i].roll_velocity,bst[i].pitch_velocity,bst[i].yaw_velocity);
```

```
    }
          len=(long)(strlen(tempmsg));


       memset(message_buffer,0x00,WIDE);

sprintf(message_buffer,"%ld,%ld,%ld,%ld,%ld,%s",from,to,cmd,val,len,tempmsg);
         if((ret=write(write_port,message_buffer,strlen(message_buffer)))<0)
           printf("APA512S There's a problem in sending data out.\n");

          if(display_data)
               printf("APA512S Message sent:\n%s\n",tempmsg);

         break;

      case SENDING_BLOBS_SHORT :

                 memset(tempmsg,0x00,WIDE);

    delta = MAXBLOBS;
for(i=0;i<=(int)(val/MAXBLOBS);i++) {
     if(i==(int)(val/MAXBLOBS))  delta = val%MAXBLOBS;
     for(j=0;j<delta;j++) {
        sprintf(&tempmsg[strlen(tempmsg)],
      "%ld,%ld,%ld,%ld,%ld,%ld,%ld,%ld,%f,%f,%f,%f,%f,%f,%f,%f,%f,",

sst[i*MAXBLOBS+j].blobnum,sst[i*MAXBLOBS+j].lblobnum,sst[i*MAXBLOBS+j].p
rocid,sst[i*MAXBLOBS+j].timestamp,

sst[i*MAXBLOBS+j].sourceid,sst[i*MAXBLOBS+j].idclass,sst[i*MAXBLOBS+j].idco
de,sst[i*MAXBLOBS+j].quality,

sst[i*MAXBLOBS+j].azm_centroid,sst[i*MAXBLOBS+j].elv_centroid,sst[i*MAXBL
OBS+j].rng_centroid,

sst[i*MAXBLOBS+j].roll_centroid,sst[i*MAXBLOBS+j].pitch_centroid,sst[i*MAXB
LOBS+j].yaw_centroid,

sst[i*MAXBLOBS+j].height_env,sst[i*MAXBLOBS+j].width_env,sst[i*MAXBLOBS+
j].depth_env);
}

          len = (long)(strlen(tempmsg));
```

```
        memset(message_buffer,0x00,WIDE);

sprintf(message_buffer,"%ld,%ld,%ld,%ld,%ld,%s",from,to,cmd,val,len,tempmsg);
          if((ret=write(write_port,message_buffer,strlen(message_buffer)))<0)
            printf("APA512S There's a problem in sending data out.\n");

           if(display_data)
                printf("APA512S Message sent:\n%s\n",tempmsg);
        }
            memset(tempmsg,0x00,WIDE);
            break;

        case    SENDING_GRASP_REGIONS :

            break;

        case    SENDING_GRASP_MONITORS :

            break;

         default :

                printf("\nAPA512S Command not understood. Try Again!!!\n");
            break;
}

}

/*********************************************************************/
/* Send variety of messages based on return values from image    */
/* processing routines.                                              */
/*********************************************************************/

send_message(ret,   no_blobs)

int  ret,  no_blobs;

{
    int      cam;

              cam = (int)bst[0].sourceid;

            if(ret<0) {
```

```
            send_return(HEALTH_STATUS,ABNORMAL);
                printf("APA512S Error snapping data into APA512 boards, and
ABNORMAL is sent.\n");
          } else if((no_blobs==0)||(ret==NO_BLOBS_FOUND)) {
              send_return(NO_BLOBS_FOUND,cam);
              printf("APA512S NO_BLOBS_FOUND is sent.\n");
         } else if (ret==TARGET_NOT_IN_SCENE) {
              send_return(TARGET_NOT_IN_SCENE,cam);
              printf("APA512S TARGET_NOT_IN_SCENE is sent.\n");
             if(no_blobs!=0) {
                  send_blob_dump(no_blobs);
                 if(display_data)
                      printf("APA512S %d blobs sent.\n",no_blobs);
            }
         } else {
               send_blob_dump(no_blobs);
              if(display_data)
                      printf("APA512S %d blobs recognized.\n",no_blobs);
         }
}
/*********************************************************************/
/* Video switch request.                                             */
/*********************************************************************/

video_switch()

{
    int      i, ret, ret1;
    long     cmd, val, len;

   cmd  = (long)(B14_VIDEO_SELECT_CHANNEL);
    val   = (long)bst[0].sourceid;

      memset(tempmsg,0x00,WIDE);

   len  = 0l;

      memset(message_buffer,0x00,WIDE);

sprintf(message_buffer,"%ld,%ld,%ld,%ld,%ld,%s",from,to,cmd,val,len,tempmsg);
        if((ret=write(write_port,message_buffer,strlen(message_buffer)))<0)
          printf("APA512S There's a problem in sending data out.\n");

    /* expect to get 0x1116 back */
```

```
read_again:
     ret1 = read_message();
     if(ret1!=IGNORE) return(ret1);
     else goto read_again;

}

/*****************************************************************************/

response()

{
   int     ret;

     switch((int)(msg.functn)) {

      case   0:
             return(IGNORE);
                    /* faulty message- something that doesn't make sense */

      case   PCIF_START_SENDING_VISUAL_RESULTS:
               return(IGNORE);  /* ignore, take no action */

      case   PCIF_STOP_SENDING_VISUAL_RESULTS:
               return(IGNORE);  /* ignore, take no action */

      case   SET_THRESHOLD_VALUE:

                threshold = (int)msg.modifer;
                if(threshold > 255) threshold = 255;
                 printf("APA512S Threshold is set to %d.\n",threshold);

               disable_boards();
               if((ret_en=enable_boards())!=SUCCESS)
                     printf("APA512S There's a problem in enabling APA512S.\n");
             else    return(RESPOND);

      case   SET_FLOOR_CUTOFF_LINE:

                floor_line = (int)msg.modifer;
               if(floor_line > 480) floor_line = 480;
                 printf("APA512S Floor_line is set to %d.\n",floor_line);
             return(RESPOND);
```

```
    case   ENABLE1:

        if((ret_en=enable_boards())!=SUCCESS)
            printf("APA512S There's a problem in enabling APA512S.\n");
      else
            printf("APA512S The APA512S has been enabled.\n");

      return(RESPOND);

    case   DISABLE1:

          if((ret=disable_boards())!=SUCCESS)
              printf("APA512S There's a problem in disabling APA512S.\n");
        else {
              printf("APA512S The APA512S has been disabled.\n");
           return(RESTART);
        }
         break;

    case   RESET1:

          if((ret=disable_boards())!=SUCCESS)
              printf("APA512S There's a problem in resetting APA512S, and
ABNORMAL is sent.\n");
        else {
             printf("APA512S The APA512S is reset.\n");
          return(RESTART);
        }
         break;

   case   ARE_YOU_HERE_TODAY :

        if(ret_en!=SUCCESS) {
           send_return(NOT_HERE,NUL);
            printf("APA512S NOT_HERE NUL is sent.\n");
        } else {
           send_return(I_AM_HERE_TODAY,NUL);
            printf("APA512S I_AM_HERE_TODAY NUL is sent.\n");
        }
         return(RESPOND);

    case   SEND_HEALTH_STATUS :
```

```
            if(ret_en!=SUCCESS) {
                send_return(HEALTH_STATUS,ABNORMAL);
                printf("APA512S HEALTH_STATUS ABNORMAL is sent.\n");
            } else {
                send_return(HEALTH_STATUS,NOMINAL);
                printf("APA512S HEALTH_STATUS NOMINAL is sent.\n");
            }
            return(RESPOND);

        case    DETERMINE_GRASP :

                send_return(NO_GRASP_REGION,(int)msg.modifer);
                printf("APA512S NO_GRASP_REGION is sent.\n");
                return(RESPOND);

        case    MONITOR_GRASP :

                send_return(NO_GRASP_REGION,(int)msg.modifer);
                printf("APA512S NO_GRASP_REGION is sent.\n");
                return(RESPOND);

        case    IDLE :

                printf("APA512S IDLE is received. No action is taken.\n");
                return(IGNORE);

        case    RESUME :

                printf("APA512S RESUME is received. No action is taken.\n");
                return(IGNORE);

        case    START_PROCESSING :

                printf("APA512S START_PROCESSING is received. No action is
taken.\n");
                return(IGNORE);

        case    SINGLE_FRAME :

                printf("APA512S SINGLE_FRAME is received. No action is
taken.\n");
                return(IGNORE);

        case    SET_TIME :
```

```
                time_diff = (int)msg.modifer - (int)time(NULL)*TICKS;
                 printf("APA512S Clock has been set to %ld ticks.\n",msg.modifer);

              return(RESPOND);

         case    B14_VIDEO_SELECT_COMPLETE : /* We receive 0x1116 here */
              return(VIDEO_SWITCHED);

       case   STOP_BLOB_TRACK:
              return(PROCESS);

      case  BLOB_DETECT_AND_TRACK:
              return(PROCESS);

       case   TARGET_ACQ_MANUAL:
              return(PROCESS);

       case   TARGET_ACQ_AUTO:
              return(PROCESS);

       case   BLOB_DETECT:
              return(PROCESS);

       case   BLOB_TRACK:
              return(PROCESS);

         default:

              return(NOT_UNDERSTOOD);  /* command not understood */
    }

}
/*************************************************************************/
check_message()

{
    int      pos, ret, ret1;

          memset(tempmsg,0x00,WIDE);

        read_again:
          if((char_count=_gs_rdy(read_port))>0)    {
                memset(message_buffer,0x00,WIDE);
```

```
                read(read_port,message_buffer,char_count);
               strcat(tempmsg,message_buffer);
            comma_count=0;
             for(pos=0;pos<strlen(tempmsg);pos++) {
                if(tempmsg[pos] == ',') comma_count++;
          }
            ret=decision();

if(ret==READ_AGAIN) goto read_again;
else  return(ret);
                    /* there better be some message waiting to be read */

} /* end of if(_gs_rdy()) */

    else return(NO_MESSAGE);

}
/*************************************************************************/

decision()

{
   int     ret, ret1;

                  memset(message_buffer,0x00,WIDE);
                   strcpy(message_buffer,tempmsg);

if ((comma_count == 5)
                   &&(message_buffer[strlen(message_buffer)-3] == ',')
                   &&(message_buffer[strlen(message_buffer)-2] == '0')
                   &&(message_buffer[strlen(message_buffer)-1] == ','))
{
        sscanf(message_buffer,"%ld,%ld,%ld,%ld,%ld,",
                 &msg.source,&msg.destin,&msg.functn,&msg.modifer,&msg.length);

              print_message();
               return(response());

} /* end of if(comma_count==5) */

else  if((comma_count==10)
                   &&(message_buffer[strlen(message_buffer)-3] == ',')
                   &&(message_buffer[strlen(message_buffer)-2] == '0')
                   &&(message_buffer[strlen(message_buffer)-1] == ','))
```

```
{ /* two messages */
    sscanf(message_buffer,"%ld,%ld,%ld,%ld,%ld,%ld,%ld,%ld,%ld,%ld,",
                &msg.source,&msg.destin,&msg.functn,&msg.modifer,&msg.length,
              &dummy1,&dummy2,&dummy3,&dummy4,&dummy5);

     printf("APA512S Message received (in Hex):
%lx,%lx,%lx,%lx,%lx,%lx,%lx,%lx,%lx,%lx,\n",
                msg.source,msg.destin,msg.functn,msg.modifer,msg.length,
                 dummy1,dummy2,dummy3,dummy4,dummy5);

       ret = response();
        if(ret==RESTART) return(RESTART); /* to handle disable,reset */
      else if(ret==PROCESS) return(PROCESS);
                    /* to handle target_man,_auto,etc.
                        second_message() takes care of second part
                     of the message */
      else if((ret==VIDEO_SWITCHED)||(ret==IGNORE))
                          /* can't ignore second part of the message */
     {
          msg.source = dummy1;
          msg.destin = dummy2;
          msg.functn = dummy3;
          msg.modifer = dummy4;
          msg.length = dummy5;

            ret1 = response();
           if((ret==IGNORE)&&(ret1==IGNORE)) return(IGNORE);
            else if(ret1==RESTART) return(RESTART);
           else if(ret1==PROCESS) return(PROCESS);
           else return(VIDEO_SWITCHED);
      } /* end of else if */
} /* end of else if(comma_count==10) */

else if(comma_count == 12) {   /* To handle BLOB_TRACK command */
       sscanf(message_buffer,"%ld,%ld,%ld,%ld,%ld,",
                &msg.source,&msg.destin,&msg.functn,&msg.modifer,&msg.length);
            print_message();
             return(response());

} /* end of else if(comma_count == 12) */

                  /* there better be some message waiting to be read */
else return(READ_AGAIN);
```

```
}

/*************************************************************************/
print_message()
{

    printf("APA512S Message received (in Hex): %lx,%lx,%lx,%lx,%lx,\n",
                msg.source,msg.destin,msg.functn,msg.modifer,msg.length);
}
/*************************************************************************/
second_message()

{
    if(dummy3>0) {
        print_message();
        msg.source = dummy1;
        msg.destin = dummy2;
        msg.functn = dummy3;
        msg.modifer = dummy4;
        msg.length = dummy5;
         return(response());
   }
}
/*************************************************************************/
```

```
/*****************************************************************
*  Init.c                                                        *
*                                                                *
*  Author  :      Alan Smith                                     *
*                 LESC, Houston, Tx.                             *
*****************************************************************/

/* Header files */

#include <stdio.h>
#include <time.h>
#include <modes.h>
#include <std_def.h>     /* Standard include file for REALib.l */

#include <vfConfig.h>
#include <kernel.h>
#include <apHead.h>
#include <blobHead.h>
#include <discrim.h>
#include <aai.h>
#include <user.h>
#include <vision.h>
#include <homemade_kernel.h>

extern int  conFlags;
extern PARAM_FLAGS  flags;
extern CONNECT_BIDS boards;
extern BLOB_DESC *bd1, *bd2, *bdtemp;
extern int  threshold;
char   key_name[NUM_HOMEMADE_KERNELS][40];
KERNEL homemade_kernel_list[NUM_HOMEMADE_KERNELS] =
{{key_name[0]}};

/*****************************************************************/
/* Initialize_boards                                             */
/*****************************************************************/

initialize_boards()

{
        startup();    /* base-level initializations */

    /** read in the hardware configuration description, initialize database
        with specific hardware information and initialize hardware **/
```

```
    if (!config("aaiboard"))
     {

      return(FAILURE);
       shutdown(1);
      exit(1);
     }

     if (!envconfig("environ"))
     {
      return(FAILURE);
       shutdown(1);
      exit(1);
     }

 /********* Image acquisition and display *********/
printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
printf("\t\t\t\tLOCKHEED ENGINEERING AND SCIENCES COMPANY\n");
printf("\t\t\t\t    APA512S IMAGE PROCESSING LABORATORY\n");
     printf("\nAPA512S Live input is being displayed.\n");
    display( "in0");        /* display live input */

    /* Set system */

     set_unsigned();

   /* Load in homemade kernels */
     if(  !kernel_load("/h0/aai/conv/homemade.ker",homemade_kernel_list)  )
      printf("Error loading kernel.\n");

return(SUCCESS);
}
/*************************************************************************
*   Enable APA512S boards                                                *
*************************************************************************/

enable_boards()

{

int             max_read = MAX_NUM_BLOBS;
int             ret;
int              maxsp_first = MAXSP_FIRST;
```

```
char            buf[5];

        /** Connect video source to APA512 board **/
   if (!apa_connect_full( "in0", VFIR_FILTERING | MAXSP_THRESHOLD |
DG_OUTPUT | OUTPUT_PROCESSED | GH_OVERLAY, threshold, 0,
homemade_kernel_list,  &boards,  &conFlags))
      return(FAILURE);
   else
    {
         /** indicate what parameters should be calculated for the APA512 **/
    flags.raw_flags = SUMX | SUMY | SUMXX | SUMYY | SUMXY | NPIXELS |
PERIMXY | MAX_MIN | RAW_PERIM;
    flags.cooked_flags = ANGLE | XY_PERIM | AREA | CENTROID | MAJOR |
MINOR | AXRATIO | XYDIFF | BOXAREA | CGDIST | RAW_PEROUND | BXARATIO;
    flags.pcinfo_flags = NHOLES | PERIMETER | TOTAL_AREA | HOLE_AREA |
HOLE_RATIO | PEROUND | PPDA;

        blob_param_check(&flags); /* assure parameter consistency */

                /** open software channel to board **/
       if ((bd1=blob_open( boards.ap_board->fd, 0, 0, READ_ALL, max_read,
&flags))
        == NULL)
       {
           printf( "APA512S Error opening software channel to APA512.\n");
        return(FAILURE);

          /** disconnect APA-512 board **/
          apa_disconnect("in0", &boards);

        /** shut down REALib.l **/
         shutdown(0);
      }

    if ((bd2=blob_open( boards.ap_board->fd, 0, 0, READ_ALL, max_read,
&flags))
        == NULL)
       {
           printf( "APA512S Error opening software channel to APA512.\n");
        return(FAILURE);

          /** disconnect APA-512 board **/
          apa_disconnect("in0", &boards);
```

```
        /** shut down REALib.l **/
         shutdown(0);
      }
   }
   return(SUCCESS);
}

/*********************************************************************
*  Disable APA512S boards                                            *
*********************************************************************/

disable_boards()

{
   int     ret;

    /** close the channel to the APA-512 **/
     blob_close(bd1);
     blob_close(bd2);

    /** disconnect APA-512 board **/
    if((ret=apa_disconnect("in0", &boards))<0) {
      return(FAILURE);
   }

return(SUCCESS);
}

/*********************************************************************
* Help Message                                                       *
*********************************************************************/

help_message()

{
    printf("\nAPA512S COMMAND LINE OPTIONS:\n");
     printf("`help' - prints out this help message.\n");
     printf("`data' - prints out message data, both coming and going.\n");
    printf("TO EXIT - Control-C should do it!\n");

}
/*********************************************************************/
```

```
/*********************************************************************
*  Track.c                                                           *
*                                                                    *
*  Author  :      Alan T. Smith                                      *
*                 LESC, Houston, TX                                  *
*********************************************************************/

/* Header files */

#include <stdio.h>
#include <time.h>
#include <modes.h>
#include <std_def.h>     /* Standard include file for REALib.l */
#include <math.h>
#include <vfConfig.h>
#include <kernel.h>
#include <apHead.h>
#include <blobHead.h>
#include <discrim.h>
#include <aai.h>
#include <user.h>
#include <vision.h>
extern   struct blob_structure bst[];
extern   struct blob_structure_short sst[];

extern BLOB_DESC *bd1, *bd2, *bdtemp;
extern CONNECT_BIDS boards;
extern PARAM_FLAGS flags;
extern int conFlags;
extern  int  display_data;
extern int floor_line;
extern int time_diff;
extern  int  target_acq_auto_flag,  frame_count;
int      i, j, l, cam, NumberOfBlobs, time_snap;
int       flag, erase_flag, astro_count, wrench_count, shuttle_count,
found_count=0,  tracking_count;
int    CAM,CLASS,CAM_INIT;
float  HGHT, WDTH, DPTH, AZM, ELV, RNG, PROB,HGHT_INIT, WDTH_INIT,
DPTH_INIT, AZM_INIT, ELV_INIT, RNG_INIT;
int      class_assign;
float    class_prob;
short    astro[BLOBS], wrench[BLOBS], shuttle[BLOBS], tracking[1];

/*********************************************************************
```

```
*                    Image_Process                                    *
*********************************************************************/

image_process(no_send)

int  *no_send;

{
    int cam = bst[0].sourceid,
       model = bst[0].idcode;

     if (!blob_snap(bd1)) /* snap the blobs into the APA-512 */
        return(NO_BLOBS_FOUND);

     /** set time for raw data acquisition **/
     time_snap = (long)time(NULL)*TICKS + (long)time_diff;

     /** read the blobs from the hardware **/
    if ((NumberOfBlobs = blob_read_all(bd1)) <= 0)  return(NO_BLOBS_FOUND);

    /* Erase old boxes */
    erase_box();
     astro_count = 0; wrench_count = 0;   shuttle_count = 0;

     /** Make blob structure--bst[blob_no]->azm_centroid = bd->raw[i].xcent */
    for(i=0,*no_send=0,flag = UNSET;i<NumberOfBlobs;i++)
   {
       if( bd1->raw[i].npixels < MAXBLOBSIZE && bd1->raw[i].npixels >
MINBLOBSIZE && (bd1->raw[i].edge_polarity == 1 || bd1->raw[i].edge_polarity
== 3) && bd1->cooked[i].ycent < floor_line)
        {    /* edge_polarity => white on black, on or off the screen */
      /*  Classify blob as ASTRONAUT, WRENCH, SHUTTLE */
       classify();

        /* Set up box drawing arrays */
       if(class_assign == ASTRONAUT)    { astro[astro_count] = i; astro_count++; }
        else if(class_assign == WRENCH){ wrench[wrench_count] = i;
wrench_count++; }
        else if(class_assign == SHUTTLE){ shuttle[shuttle_count] = i;
shuttle_count++; }

       if(target_acq_auto_flag == SET && class_assign == model)     flag = SET;

      if( class_assign != NEITHER )
```

```
            bst[*no_send].idclass = sst[*no_send].idclass = (long)IDCLASS_KNOWN;
        else if (class_assign == NEITHER)
            bst[*no_send].idclass = sst[*no_send].idclass =
(long)IDCLASS_OBSTACLE;

        if(target_acq_auto_flag == UNSET || (target_acq_auto_flag == SET &&
class_assign != NEITHER) )
      {
            bst[*no_send].idcode = sst[*no_send].idcode = (long)class_assign;
            bst[*no_send].quality = sst[*no_send].quality = (long)class_prob;
            bst[*no_send].blobnum = sst[*no_send].blobnum = (long)(i+1);
            bst[*no_send].lblobnum = sst[*no_send].lblobnum = (long)(i+1);
            bst[*no_send].procid = sst[*no_send].procid = (long)(0x50e);
            bst[*no_send].timestamp = sst[*no_send].timestamp =
(long)(time_snap)ntroid = SCALE_YCENT*bd1->cooked[i].ycent;
            bst[*no_send].rng_centroid = sst[*no_send].rng_centroid =
(float)NULL_DATA_FLOAT;
            bst[*no_send].roll_centroid = sst[*no_send].roll_centroid =
NULL_DATA_FLOAT;
            bst[*no_send].pitch_centroid = sst[*no_send].pitch_centroid =
NULL_DATA_FLOAT;
            bst[*no_send].yaw_centroid = sst[*no_send].yaw_centroid =
NULL_DATA_FLOAT;
            bst[*no_send].height_env = sst[*no_send].height_env = bd1-
>cooked[i].ydiff;
            bst[*no_send].width_env = sst[*no_send].width_env = bd1-
>cooked[i].xdiff;
            bst[*no_send].depth_env = sst[*no_send].depth_env =
NULL_DATA_FLOAT;
            bst[*no_send].azm_env_center = (bd1->raw[i].maxx + bd1-
>raw[i].minx)/2;
          bst[*no_send].elv_env_center = SCALE*SCALE_YCENT*(bd1-
>raw[i].maxy + bd1->raw[i].miny)/2;
           bst[*no_send].rng_env_center = NULL_DATA_FLOAT;
          bst[*no_send].azm_velocity = NULL_DATA_FLOAT;
          bst[*no_send].elv_velocity = NULL_DATA_FLOAT;
          bst[*no_send].rng_velocity = NULL_DATA_FLOAT;
          bst[*no_send].roll_velocity = NULL_DATA_FLOAT;
          bst[*no_send].pitch_velocity = NULL_DATA_FLOAT;
          bst[*no_send].yaw_velocity = NULL_DATA_FLOAT;

         printf("APA512S Object %d: (%.2lf,%.2lf)
Npixels:%d\n",class_assign,bd1->cooked[i].xcent,
            SCALE_YCENT*bd1->cooked[i].ycent,bd1->raw[i].npixels);
```

```
            *no_send += 1;
        }
    }   /* END size and color */
    }   /* END for #blobs */

        /* Draw new boxes */
        draw_box();

        if(target_acq_auto_flag == SET && flag == UNSET)
                return(TARGET_NOT_IN_SCENE);
        else            return(SUCCESS);
}

/*************************************************************************
*                       Blob_Track                                       *
*************************************************************************

This program sends information for three types of blobs:
        Blob being tracked, Recognized objects, All other blobs(obstacles).
The code was requested to be written in three modules in case B. Goode
changed
his mind on what to send. */

blob_track(no_send,cam)

int   *no_send, *cam;

{
    int       in_range_flag, blob_found_flag, found[BLOBS], apa_tracked_blob,
tracked_blob;
    double   dist, greater_dist;

    if(frame_count == 0)     /* Record initial parameters */
  {
        ELV = bst[0].elv_centroid;
        AZM = bst[0].azm_centroid;
        RNG = bst[0].rng_centroid;
        HGHT = bst[0].height_env;
        WDTH = bst[0].width_env;
        DPTH = bst[0].depth_env;
        CAM = *cam = (int)bst[0].sourceid;
        printf("Search params:(x,y)=(%.1f,%.1f) WDTH=%.1f
HGHT=%.1f\n",AZM,ELV,WDTH,HGHT);
  }
```

```
    frame_count++;

      printf("\nAPA512S  Frame#  %d\n",frame_count);

    memset(bst,0x00,BLOBS);
    memset(sst,0x00,BLOBS);

     if (!blob_snap(bd1)) /* snap the blobs into the APA-512 */
   return(NO_BLOBS_FOUND);

    /** set time for raw data acquisition **/
    time_snap = (long)time(NULL)*TICKS + (long)time_diff;

    /** read the blobs from the hardware **/
    if ((NumberOfBlobs = blob_read_all(bd1)) <= 0)  return(NO_BLOBS_FOUND);

    /* Erase old boxes */
    erase_box();

    /* Initialize blob draw counters */
    astro_count = 0;  wrench_count = 0; shuttle_count = 0;  tracking_count = 0;

    /** First module- find target being tracked **/

     for(i=0,*no_send=0,found_count=0,blob_found_flag = UNSET,tracked_blob =
UNSET; i<NumberOfBlobs; i++)
   {
       if( bd1->raw[i].npixels < MAXBLOBSIZE && bd1->raw[i].npixels >
MINBLOBSIZE
         && (bd1->raw[i].edge_polarity == 1 || bd1->raw[i].edge_polarity == 3) &&
bd1->cooked[i].ycent < floor_line) /* Size, color, and above floor noise */
        {   /* edge_polarity => white on black, on or off the screen */
          if(bd1->cooked[i].xcent-CENT_VAR < AZM && AZM < bd1-
>cooked[i].xcent+CENT_VAR
          && bd1->cooked[i].ycent-CENT_VAR < ELV && ELV < bd1-
>cooked[i].ycent+CENT_VAR
          && bd1->cooked[i].xdiff-DIFF_VAR < WDTH && WDTH < bd1-
>cooked[i].xdiff+DIFF_VAR
          && bd1->cooked[i].ydiff-DIFF_VAR < HGHT && HGHT < bd1-
>cooked[i].ydiff+DIFF_VAR)
            {   tracking_count = 1; /* Used for boxing routine */
              found[found_count] = i;
               found_count++;  /* Calculate # blobs found */
```

```
        }
      }
    }  /* End for i loop */
printf("found_count  frame  %d=%d\n",frame_count,found_count);
    if(frame_count == 1 && found_count > 1) /*More than one matched in
frame  1*/
        for(greater_dist=0.0,i=0;i<found_count;i++) /* Calculate distance */
      {
            dist = pow(bd1->cooked[found[i]].xcent-AZM,2.0) + pow(bd1-
>cooked[found[i]].ycent-ELV,2.0);
          if(dist > greater_dist) {   greater_dist = dist;
             apa_tracked_blob = found[i];    tracked_blob = *no_send;    }
      }
    else if(found_count > 0)    {   apa_tracked_blob = found[0];
          tracked_blob = *no_send;    }   /* Else pick the first */
/*printf("tracked_blob=%d\n",tracked_blob);
*/  if(found_count > 0)
   {
      blob_found_flag = SET;  /* Use for TARGET_NOT_IN_SCENE */

       if(frame_count == 1)
        {   /* Recognize if in first frame-new blob to track */
           i = apa_tracked_blob;   /* Needed for bd1->[i] in classify() */

         classify();

         if(class_assign != NEITHER)
              bst[*no_send].idclass = sst[*no_send].idclass =
(long)IDCLASS_KNOWN;
          else if(class_assign == NEITHER)
              bst[*no_send].idclass = sst[*no_send].idclass =
(long)IDCLASS_TARGET1;

            bst[*no_send].idcode = sst[*no_send].idcode = class_assign;
            bst[*no_send].quality = sst[*no_send].quality = class_prob;
          bst[*no_send].azm_velocity = NULL_DATA_FLOAT;
printf("1:azm_velocity=%.3lf\n",bst[*no_send].azm_velocity);
          bst[*no_send].elv_velocity = NULL_DATA_FLOAT;
      }
       else if(frame_count >1)
        {   /* Params from last frame -- compute velocities */
          bst[*no_send].idcode = sst[*no_send].idcode = CLASS;
           bst[*no_send].quality = sst[*no_send].quality = PROB;
```

```
            bst[*no_send].azm_velocity = bd1->cooked[apa_tracked_blob].xcent -
AZM;
printf("2:azm_velocity=%.3lf\n",bst[*no_send].azm_velocity);
            bst[*no_send].elv_velocity = bd1->cooked[apa_tracked_blob].ycent -
ELV;
      }
        /* Fill in rest of structure */
         bst[*no_send].blobnum = sst[*no_send].blobnum = (long)(*no_send+1);
         bst[*no_send].lblobnum = sst[*no_send].lblobnum = (long)(*no_send+1);
        bst[*no_send].procid = sst[*no_send].procid = (long)0x50e;
         bst[*no_send].timestamp = sst[*no_send].timestamp = (long)time_snap;
        bst[*no_send].sourceid = sst[*no_send].sourceid = (long)(*cam) =
(long)CAM;
         bst[*no_send].azm_centroid = sst[*no_send].azm_centroid = bd1-
>cooked[apa_tracked_blob].xcent;
         bst[*no_send].elv_centroid = sst[*no_send].elv_centroid =
SCALE_YCENT*bd1->cooked[apa_tracked_blob].ycent;
         bst[*no_send].rng_centroid = sst[*no_send].rng_centroid =
NULL_DATA_FLOAT;
         bst[*no_send].roll_centroid = sst[*no_send].roll_centroid =
NULL_DATA_FLOAT;
         bst[*no_send].pitch_centroid = sst[*no_send].pitch_centroid =
NULL_DATA_FLOAT;
         bst[*no_send].yaw_centroid = sst[*no_send].yaw_centroid =
NULL_DATA_FLOAT;
         bst[*no_send].height_env = sst[*no_send].height_env = bd1-
>cooked[apa_tracked_blob].ydiff;
         bst[*no_send].width_env = sst[*no_send].width_env = bd1-
>cooked[apa_tracked_blob].xdiff;
         bst[*no_send].depth_env = sst[*no_send].depth_env =
NULL_DATA_FLOAT;
          bst[*no_send].azm_env_center = (bd1->raw[tracked_blob].maxx + bd1-
>raw[apa_tracked_blob].minx)/2;
        bst[*no_send].elv_env_center =  SCALE*SCALE_YCENT*(bd1-
>raw[apa_tracked_blob].maxy  +  bd1->raw[apa_tracked_blob].miny)/2;
         bst[*no_send].rng_env_center = NULL_DATA_FLOAT;
        bst[*no_send].rng_velocity = NULL_DATA_FLOAT;
        bst[*no_send].roll_velocity = NULL_DATA_FLOAT;
        bst[*no_send].pitch_velocity = NULL_DATA_FLOAT;
        bst[*no_send].yaw_velocity = NULL_DATA_FLOAT;

         /* Set parameters for next frame */
        AZM     = bst[*no_send].azm_centroid;
        ELV     = bst[*no_send].elv_centroid;
```

```
        WDTH    = bst[*no_send].width_env;
        HGHT    = bst[*no_send].height_env;
        CLASS   = bst[*no_send].idcode;
        PROB    = bst[*no_send].quality;
        CAM     = bst[*no_send].sourceid;

         /* Set up blob draw array for tracked blob */
         tracking[0] = (short)apa_tracked_blob;

         printf("Tracked APA512S Blob: (%.2lf,%.2lf) Npixels:%d\n",bd1-
>cooked[apa_tracked_blob].xcent,SCALE_YCENT*bd1-
>cooked[apa_tracked_blob].ycent,bd1->raw[apa_tracked_blob].npixels);
       *no_send += 1;
   }

    /** 2nd & 3rd modules- recognize other objects and targets **/
   for(i=0; i<NumberOfBlobs; i++)
   {
       if( bd1->raw[i].npixels < MAXBLOBSIZE && bd1->raw[i].npixels >
MINBLOBSIZE
        && (bd1->raw[i].edge_polarity == 1 || bd1->raw[i].edge_polarity == 3) &&
i != apa_tracked_blob && bd1->cooked[i].ycent < floor_line)
        {   /* edge_polarity => white on black, on or off the screen */
          classify();
           if( class_assign == NEITHER )   /* Make structure */
           {   bst[*no_send].idclass = sst[*no_send].idclass =
(long)IDCLASS_OBSTACLE;
               bst[*no_send].idcode = sst[*no_send].idcode = 0x0;
        }
          else if( class_assign != NEITHER )
        {
               bst[*no_send].idclass = sst[*no_send].idclass =
(long)IDCLASS_KNOWN;
               bst[*no_send].idcode = sst[*no_send].idcode = class_assign;
        }
            bst[*no_send].quality = sst[*no_send].quality = class_prob;
            bst[*no_send].blobnum = sst[*no_send].blobnum = (long)(*no_send+1);
           bst[*no_send].lblobnum = sst[*no_send].lblobnum =
(long)(*no_send+1);
           bst[*no_send].procid = sst[*no_send].procid = (long)0x50e;
            bst[*no_send].timestamp = sst[*no_send].timestamp =
(long)time_snap;
           bst[*no_send].sourceid = sst[*no_send].sourceid = (long)(*cam) =
(long)CAM;
```

```
            bst[*no_send].azm_centroid = sst[*no_send].azm_centroid = bd1-
>cooked[i].xcent;
            bst[*no_send].elv_centroid = sst[*no_send].elv_centroid =
SCALE_YCENT*bd1->cooked[i].ycent;
            bst[*no_send].rng_centroid = sst[*no_send].rng_centroid =
NULL_DATA_FLOAT;
            bst[*no_send].roll_centroid = sst[*no_send].roll_centroid =
NULL_DATA_FLOAT;
            bst[*no_send].pitch_centroid = sst[*no_send].pitch_centroid =
NULL_DATA_FLOAT;
            bst[*no_send].yaw_centroid = sst[*no_send].yaw_centroid =
NULL_DATA_FLOAT;
            bst[*no_send].height_env = sst[*no_send].height_env = bd1-
>cooked[i].ydiff;
            bst[*no_send].width_env = sst[*no_send].width_env = bd1-
>cooked[i].xdiff;
            bst[*no_send].depth_env = sst[*no_send].depth_env =
NULL_DATA_FLOAT;
            bst[*no_send].azm_env_center = (bd1->raw[i].maxx + bd1-
>raw[i].minx)/2;
          bst[*no_send].elv_env_center =  SCALE*SCALE_YCENT*(bd1-
>raw[i].maxy  +  bd1->raw[i].miny)/2;
           bst[*no_send].rng_env_center = NULL_DATA_FLOAT;
          bst[*no_send].azm_velocity = NULL_DATA_FLOAT;
/*printf("3:azm_velocity=%.3lf\n",bst[*no_send].azm_velocity);
*/            bst[*no_send].elv_velocity = NULL_DATA_FLOAT;
          bst[*no_send].rng_velocity = NULL_DATA_FLOAT;
          bst[*no_send].roll_velocity = NULL_DATA_FLOAT;
          bst[*no_send].pitch_velocity = NULL_DATA_FLOAT;
          bst[*no_send].yaw_velocity = NULL_DATA_FLOAT;

            /* Set up blob draw arrays if nothing is being tracked */
          if(found_count == 0)
        {
             if(bst[*no_send].idcode == ASTRONAUT)   {   astro[astro_count] = i;
astro_count++; }
              else if(bst[*no_send].idcode == WRENCH) {   wrench[wrench_count]
= i;   wrench_count++; }
              else if(bst[*no_send].idcode == SHUTTLE) {  shuttle[shuttle_count] =
i; shuttle_count++; }
        }
         *no_send += 1;

    }
```

```
    }   /* End for i loop */

    /* Draw new boxes */
    draw_box();
/*printf("tracked_blob  at  end=%d\n",tracked_blob);
*/
    if(found_count > 0)
      {     printf("azm_velocity=%.3lf\n",bst[tracked_blob].azm_velocity);
        if(bst[tracked_blob].azm_centroid > 512-X_BORDER &&
bst[tracked_blob].azm_velocity  >  1.0)
        {   /* +/- 1.0 is approx. max. velocity due to noise */
         if(CAM == CAMERA_LEFT || CAM == CAMERA_CENTER)  /* Coordinate
transform  */
              AZM = X_BORDER - 512.0 + bst[tracked_blob].azm_centroid;

        if(CAM == CAMERA_LEFT)  *cam = CAM = CAMERA_CENTER;
        else if (CAM == CAMERA_CENTER)  *cam = CAM = CAMERA_RIGHT;
            printf("APA512S Blob moving to right -> Video switched to camera#
%d.\n",*cam);
      }

       if(bst[tracked_blob].azm_centroid < X_BORDER &&
bst[tracked_blob].azm_velocity  <  -1.0  &&  bst[tracked_blob].azm_velocity  !=
NULL_DATA_FLOAT )
      {
         if(CAM == CAMERA_CENTER || CAM == CAMERA_RIGHT) /* Coordinate
transform  */
              AZM = 512.0 - X_BORDER + bst[tracked_blob].azm_centroid;

        if(CAM == CAMERA_RIGHT)  *cam = CAM = CAMERA_CENTER;
        else if (CAM == CAMERA_CENTER)  *cam = CAM = CAMERA_LEFT;
            printf("APA512S Blob moving to left <- Video switched to camera#
%d.\n",*cam);
      }
   }

   if(blob_found_flag == UNSET)    return(TARGET_NOT_IN_SCENE);
   else                        return(SUCCESS);
}


/*************************************************************************
*   Pixel->Radian Conversion:                                            *
*          cameras 1-3              Fairchild az=.0023108(pixel#-256)     *
```

```
*                                    el=.0019864(pixel#-121)                  *
*          cameras 4-6             ELMO      az=.0015748(pixel#-256)        *
*                                    el=.0013162(pixel#-121)                  *
*****************************************************************************/

pixel_to_radian(no_send)

int  no_send;

{
   int     i;
    double   scale_x, scale_y;

     scale_x = FOV_X*(3.14159/180.0)/RES_X;
     scale_y = FOV_Y*(3.14159/180.0)/RES_Y;

    for(i=0; i< no_send; i++)
   {
        bst[i].azm_centroid = sst[i].azm_centroid        =
scale_x*(bst[i].azm_centroid-RES_X/2.0);
         bst[i].elv_centroid = sst[i].elv_centroid        = scale_y*(bst[i].elv_centroid-
RES_Y/2.0);
        bst[i].height_env    = sst[i].height_env         = scale_y*bst[i].height_env;
        bst[i].width_env     = sst[i].width_env          = scale_x*bst[i].width_env;
          bst[i].azm_env_center    = scale_x*(bst[i].azm_env_center-RES_X/2.0);
          bst[i].elv_env_center    = scale_y*(bst[i].elv_env_center-RES_Y/2.0);

        if((int)(10*bst[i].azm_velocity) != (int)(10*NULL_DATA_FLOAT))  /*
Send velocity only if tracking */
      {
          (float)bst[i].azm_velocity      *= scale_x;
          (float)bst[i].elv_velocity      *= scale_y;
      }
   }
}

/*****************************************************************************
*      Radian->Pixel Conversion:                                            *
*          cameras 1-3             Fairchild pixel# = az/0.0023108+256      *
*                                    pixel# = el/0.0019864+121                *
*          cameras 4-6             ELMO      pixel# = az/0.0015748+256      *
*                                    pixel# = el/0.0013162+121                *
*****************************************************************************/
radian_to_pixel()
```

```
{
    double  scale_x, scale_y;

    scale_x = FOV_X*(3.14159/180.0)/RES_X;
    scale_y = FOV_Y*(3.14159/180.0)/RES_Y;

    bst[0].azm_centroid      = bst[0].azm_centroid/scale_x + RES_X/2.0;
    bst[0].elv_centroid      = bst[0].elv_centroid/scale_y + RES_Y/2.0;
    bst[0].height_env        /= scale_y;
    bst[0].width_env         /= scale_x;
}

erase_box()

{
    /* Erase old boxes if needed */
   if(erase_flag == SET)
   {

blob_draw_bindex(bd2,tracking,tracking_count,BOUNDING_BOX,boards.gh_bo
ard->fd);

blob_draw_bindex(bd2,tracking,tracking_count,CENTROID_CROSS,boards.gh_b
oard->fd);
      blob_draw_bindex(bd2,astro,astro_count,BOUNDING_BOX,boards.gh_board-
>fd);

blob_draw_bindex(bd2,wrench,wrench_count,CENTROID_CROSS,boards.gh_bo
ard->fd);

blob_draw_bindex(bd2,shuttle,shuttle_count,BOUNDING_BOX,boards.gh_board
->fd);

blob_draw_bindex(bd2,shuttle,shuttle_count,CENTROID_CROSS,boards.gh_boa
rd->fd);
   erase_flag = UNSET;
   }
}

draw_box()
{

    /* Draw new boxes */
```

```
    if( found_count > 0)     /* Boxes for track() */
    {   erase_flag = SET;

blob_draw_bindex(bd1,tracking,tracking_count,BOUNDING_BOX,boards.gh_bo
ard->fd);

blob_draw_bindex(bd1,tracking,tracking_count,CENTROID_CROSS,boards.gh_b
oard->fd);
    }
     if( astro_count > 0 || wrench_count > 0 || shuttle_count > 0) erase_flag = SET;
       blob_draw_bindex(bd1,astro,astro_count,BOUNDING_BOX,boards.gh_board-
>fd);

blob_draw_bindex(bd1,wrench,wrench_count,CENTROID_CROSS,boards.gh_bo
ard->fd);

blob_draw_bindex(bd1,shuttle,shuttle_count,BOUNDING_BOX,boards.gh_board
->fd);

blob_draw_bindex(bd1,shuttle,shuttle_count,CENTROID_CROSS,boards.gh_boa
rd->fd);

     bdtemp = bd1;
     bd1 = bd2;
     bd2 = bdtemp;

}

classify()
{
FILE     *f_weights;
int      ii, jj;
double  wi[INPUTS+1][HIDDEN], wj[HIDDEN+1][CLASSES], wsumi[HIDDEN],
        wsumj[CLASSES], y_pr[HIDDEN+1], y[CLASSES], betai, betaj;

     if((f_weights = fopen("nn_weights","r"))!=NULL) {

      for(jj=0;jj<HIDDEN;jj++) {
          for(ii=0;ii<INPUTS+1;ii++) {
               fscanf(f_weights,"%lf",&wi[ii][jj]);
          }
      }
      for(jj=0;jj<CLASSES;jj++) {
         for(ii=0;ii<HIDDEN+1;ii++) {
```

```
                fscanf(f_weights,"%lf",&wj[ii][jj]);
            }
        }
          fclose(f_weights);
    } else {
          printf("\nCan't open the file.\n");
    }

     /* Nonlinearity steepness */
     betai = 100.0;  betaj = 10.0;

     /* Calculate output of first layer */
   for(ii=0;ii<HIDDEN;ii++)
   {
     y_pr[ii] = bd1->cooked[i].axratio*wi[0][ii] +
          bd1->pcinfo[i].peround*wi[1][ii] - wi[INPUTS][ii]; /* Weighted sum */
     y_pr[ii] = 1/(1+exp(-betai*y_pr[ii]));                  /* Nonlinearity */
   }
        /* Calculate output of second layer */
   for(jj=0;jj<CLASSES;jj++)
   {
       y[jj]=0.0;
       for(ii=0;ii<HIDDEN;ii++)    y[jj] += y_pr[ii]*wj[ii][jj];
        y[jj] -= wj[HIDDEN][jj];              /* Subtract threshold */
         y[jj] = 1/(1+exp(-betaj*y[jj]));     /* Nonlinearity */
/*       printf("y[%d]=%lf\n",jj,y[jj]);
*/  }

   if(y[0]>.5 && y[1]<.5 && y[2]<.5)
   {
       class_assign = ASTRONAUT;
        class_prob = 100.0*y[0];
   }
   else if(y[0]<.5 && y[1]>.5 && y[2]<.5)
   {
       class_assign = WRENCH;
        class_prob = 100.0*y[1];
   }
   else if(y[0]<.5 && y[1]<.5 && y[2]>.5)
   {
       class_assign = SHUTTLE;
        class_prob = 100.0*y[2];
   }
   else
```

```
    {
        class_assign = NEITHER;
         class_prob = 99.9;
    }
}

/*****************************************************************/
```

```
#define MODIFER
#define PARM
#define VISION_CTL                          0x500
#define PCIF                                0x501
#define DISPLAY                             0x502
#define VIDEO_CTL                           0x503
#define RANGING_CTL                         0x504
#define JUDAY_VIDEO                         0x50B
#define APA512S                             0x50E

#define WORLD_MODEL                         0x600

#define ENABLE1                             0x1000
#define DISABLE1                            0x1001
#define RESET1                              0x1002
#define SAFE                                0x1003
#define IDLE                                0x1004
#define RESUME                              0x1005
#define START_PROCESSING                    0x1007
#define SINGLE_FRAME                        0x1008
#define DEBUG_ON                            0x1009
#define DEBUG_OFF                           0x100A
#define SET_TIME MODIFER                    0x100B
#define TARGET_ACQ_MANUAL  MODIFER          0x100C
#define TARGET_ACQ_AUTO MODIFER             0x100D
#define BLOB_DETECT_AND_TRACK               0x100E
#define BLOB_DETECT                         0x100F
#define BLOB_TRACK PARM                     0x1010
#define DETERMINE_GRASP                     0x1011
#define MONITOR_GRASP                       0x1012
#define TARGET_CONFIRMED                    0x1013
#define TARGET_NOT_IN_SCENE                 0x1014
#define ARE_YOU_HERE_TODAY                  0x1015
#define SEND_HEALTH_STATUS                  0x1016
#define BLOB_DUMP_REQUEST MODIFER           0x1017
#define SWITCH_TO_INTERNAL_FEATURE          0x1018
#define STOP_BLOB_TRACK                     0x1019
#define SET_THRESHOLD_VALUE                 0x101A
#define SET_FLOOR_CUTOFF_LINE               0x1024
#define SENDING_BLOBS MODIFER               0x1100
#define SENDING_BLOBS_SHORT MODIFER         0x1101
#define SENDING_GRASP_REGIONS MODIFER       0x1102
#define SENDING_GRASP_MONITORS MODIFER      0x1103
#define NO_BLOBS_FOUND                      0x1108
```

```
#define NO_GRASP_REGION                          0x1109
#define HEALTH_STATUS                            0x110A
#define I_GIVE_UP                                0x110B
#define I_AM_HERE_TODAY                          0x110C
#define WARNING_TARGET_TOO_BIG                   0x110E
#define SENDING_RAW_MAPPER_DATA MODIFER          0x110F
#define SENDING_RAW_VIDEO_DATA MODIFER           0x1110
#define DEBUG_MESSAGE                            0x1111
#define VISION_SYSTEM_HEALTH_STATUS              0x1112
#define B14_VIDEO_SELECT_CHANNEL MODIFER         0x1114
#define WORLD_MODEL_ALIVE_TODAY                  0x1115
#define B14_VIDEO_SELECT_COMPLETE                0x1116
#define PCIF_START_SENDING_VISUAL_RESULTS        0x1509
#define PCIF_STOP_SENDING_VISUAL_RESULTS         0x150A

#define CAMERA_LEFT                              1
#define CAMERA_CENTER                            2
#define CAMERA_RIGHT                             3
#define CAMERA_HEAD                              4
#define CAMERA_RHAND                             5
#define CAMERA_LHAND                             6
#define CAMERA_MAPPER_VIDEO                      7
#define MAPPER                                   10
#define NOMINAL                                  1
#define ABNORMAL                                 -1
#define NOT_HERE                                 2
#define NULL_DATA_INT                            -991
#define NULL_DATA_FLOAT                          -99.9
#define NUL                                      0
#define VIDEO_SWITCHED                           2
#define IGNORE                                   1
#define RESPOND                                  1

#define ASTRONAUT                                1
#define WRENCH                                   2
#define SHUTTLE                                  3
#define NEITHER                                  0

#define IDCLASS_OBSTACLE                         3
#define IDCLASS_TARGET1                          1
#define IDCLASS_TARGET2                          2
#define IDCLASS_KNOWN                            4
#define GRASPCLASS_UNKNOWN                       5
#define GRASPCLASS_KNOWN                         6
```

```
#define MODELCLASS_OBSTACLE                7
#define MODELCLASS_TARGET                  8
#define LEFT_HAND                          9
#define RIGHT_HAND                         10
#define ONE_ARM                            11
#define TWO_ARM                            12
#define CLASSES                            3

#define IO_COM                             "/t3"

#define SUCCESS                            0
#define FAILURE                            -1
#define PROCESS                            0
#define RESTART                            -1
#define NO_MESSAGE                         -100
#define NOT_UNDERSTOOD                     -5
#define READ_AGAIN                         -3
#define WIDE                               10000
#define BLOBS                              256
#define MAXBLOBS                           33
#define MINUTE                             60
#define MAXBLOBSIZE                        100000
#define MINBLOBSIZE                        50
#define TICKS                              15625
#define SCALE                              1.625
#define SCALE_YCENT                        1.23
                /* Factor to read 485 lines */
#define CENT_VAR                           70
#define DIFF_VAR                           70
#define X_BORDER                           85
#define FOV_X                              46.2
#define FOV_Y                              36.2
#define RES_X                              512
#define RES_Y                              485

#define HIDDEN                             8
#define INPUTS                             2
#define SET                                1
#define UNSET                              -1

struct blob_structure {
    int     blobnum;
    int     lblobnum;
    int     procid;
```

```
    int      timestamp;
    int      sourceid;
    int      idclass;
    int      idcode;
    int      quality;
     float azm_centroid;
     float elv_centroid;
     float rng_centroid;
     float roll_centroid;
     float pitch_centroid;
     float yaw_centroid;
     float height_env;
     float width_env;
     float depth_env;
      float azm_env_center;
      float elv_env_center;
      float rng_env_center;
     float azm_velocity;
     float elv_velocity;
     float rng_velocity;
     float roll_velocity;
     float pitch_velocity;
     float yaw_velocity;
};

struct  blob_structure_short  {
     int blobnum;
     int lblobnum;
    int procid;
     int timestamp;
    int sourceid;
    int idclass;
    int idcode;
     int quality;
     float azm_centroid;
     float elv_centroid;
     float rng_centroid;
     float roll_centroid;
     float pitch_centroid;
     float yaw_centroid;
     float height_env;
     float width_env;
     float depth_env;
};
```

```
typedef    struct
{
    long     source;
    long     destin;
    long     functn;
    long     modifer;
    long     length;
    char     *message;
} Message;
```

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
JSC-24630		

4. Title and Subtitle	5. Report Date
Vision Tracking System for the Extravehicular Activity Retriever Robot	January 1991
	6. Performing Organization Code EE

7. Author(s)	8. Performing Organization Report No.
A. T. Smith and S. Fotedar Lockheed Engineering & Sciences Company	LESC-28763
	10. Work Unit No.

9. Performing Organization Name and Address	11. Contract or Grant No.
Lockheed Engineering & Sciences Company 2400 NASA Road 1 Houston, Texas 77058-3711	NAS 9-17900
	13. Type of Report and Period Covered Report

12. Sponsoring Agency Name and Address	14. Sponsoring Agency Code
National Aeronautics and Space Administration Lyndon B. Johnson Space Center Houston, Texas 77258-8561 Technical Monitor: T. E. Fisher	

15. Supplementary Notes

16. Abstract

This report describes the application of the Adaptive Automation, Inc., machine vision processor, model APA512S, for vision tracking support to the Extravehicular Activity (EVA) Retriever (EVAR) robot. This vision tracking system provides performance superior to the McDonnell Douglas Astronautics Company (McDAC) vision tracking system presently being used.

Target recognition, target tracking, and input/output (I/O) communications interface software are presented. Target recognition is accomplished with an artificial neural network. The communications interface software sends the processed vision information from the APA512S, located in building 14 at the Lyndon B. Johnson Space Center (JSC), to the central EVAR data collection point, the vision subsystem (VISION_CTL), located in building 9A

17. Key Words (Suggested by Author(s))	18. Distribution Statement
	Unclassified - Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price*
Unclassified	Unclassified	115	

*For sale by the National Technical Information Service, Springfield, Virginia 22161