THE NBS VISION SYSTEM IN THE AMRF

NBSIR 87-3684

December 3, 1987

By:
Marilyn Nashman
Karen J. Chaconas

U.S. DEPARTMENT OF COMMERCE National Bureau of Standards Gaithersburg, Maryland

THE NBS VISION SYSTEM IN THE AMRF

Marilyn Nashman
Karen J. Chaconas

December 1987

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

This document describes the NBS Vision System in the AMRF. It discusses the objectives of the vision system and its applications in the factory environment. Since the vision system is a multi-processor system, each process is described according to its position in the vision hierarchy as well as to its particular logical and computational functions. The unique hardware used is discussed and its capabilities described. In addition, a guide to operations is included: this contains step-by-step directions for "bringing up" the system in either stand-alone mode or integrated mode. The interfaces between the individual processes of the vision system, as well as the interfaces between the vision system and other AMRF systems, are described. Finally, appendices are included which describe data structures, and debugging features.

The document may be referenced at different levels. Section II describes the vision system in very general terms; it is intended for the reader who requires only an overview of the function of vision in the AMRF. Sections III and V contain a more detailed description of the vision system and its interfaces with other systems. It assumes that the reader is familiar with computers and the concept of hierarchical design. A background in image processing techniques is useful in understanding the subroutines described. Section IV serves as an operations manual and is intended for the user who will be "bringing up" and monitoring the vision system. It assumes that the user is familiar with computers.

AMRF documentation packages describing the Horizontal Workstation and the Material Handling Workstation are useful references when reading this document.

II. SYSTEM OVERVIEW

The vision system developed by the Sensory-Interactive Robotics Group of the Robot Systems Division at NBS serves two functions in the AMRF: it is used both by the Horizontal Workstation (HWS) and the Material Handling Workstation (MHWS). In both AMRF applications, the vision system processes an image only upon request. However it is constantly polling for command requests and monitoring the status of individual vision processing modules.

1. VISION AND THE REAL-TIME CONTROL SYSTEM (RCS)

There are currently three commands defined in the interface between the vision system and the RCS. Upon receiving an RCS command, vision first determines which information is being requested. Only information relevant to the type of question being asked is returned. Question one is a request for the centroid of the object in view; this enables RCS to center the robot tool over the part. Question two is a request for verification of a specific part, and if confirmed, the position and orientation of that part. Question three is a request for the computed range to the surface of the object in view. The answers supplied by the vision system enable RCS to compute the required robot motion to pick up the part at the ideal grasp point.

The vision system always supplies an answer to RCS after a request is made. The answer includes not only the requested information but a status flag which indicates success or failure of the action. A failed action can be the result of non-recognition of an expected part or a result of a "poor" image. Causes of poor images include improper lighting conditions, camera exposure problems, or degraded camera images. As will be explained further, the vision system attempts to adjust its internal parameters to overcome failures caused by poor images. An action is considered to be successful if an expected part can be favorably matched to its internal model (see description of vision process MLDSERV), and if that part is located in the approximate portion of the tray in which it had been predicted.

2. VISION AND THE MATERIAL HANDLING WORKSTATION (MHWS)

The vision system acts to verify the contents of trays for the Material Handling Workstation. There is one command defined in the vision system's interface with MHWS. This question requests

verification, position, and orientation of parts on a tray. At present, there is only one configuration of the tray that can be verified. This request can be initiated either by an MHWS network communication or by a manually triggered signal sent from a remote location. It is expected that the methods used in the default verification can be expanded to be used on any tray configuration when actual MHWS data is entered into the AMRF database. At that time, the vision system will request a tray definition report and a tray contents report and will extract the pertinent information required for it to verify the identity and position of the expected parts. The interface between the vision system and the AMRF database has been independently tested successfully. At the completion of the verification task, vision sends a status report to MHWS. Currently, this report consists of a "done" flag, but in the future will include fields indicating success or failure of the task as well as updated positions and orientations of the verified parts.

III. ARCHITECTURE DESCRIPTION

The vision system is designed in a hierarchical manner: commands from the control system and/or the MHWS are decomposed into lower level tasks and executed by the appropriate process. There are currently six independent processes, operating asynchronously, that analyze and extract information from an image scene. Each process resides on its own microprocessor board and communicates to other processes via a pre-defined memory block (common memory). System routines have been developed to insure the integrity of this data transfer.

Currently, three of the processes operate in a bottom-up mode to read an image and globally extract information from it. These processes are First Stage Vision (FSV), Second Stage Vision A (SSVA) and Second Stage Vision B (SSVB). The Multi-Level Database Processor (MLDSERV) acts in a top-down mode using its model database to identify specific objects in a scene. The Supervisor (SUP) process monitors vision system activity and communicates with the "outside world" via the Netboard (NET) processor (used for communicating with MHWS and the AMRF Database) or a 589 board 1 (used for communicating with the RCS). Requests for information are passed from SUP to the server which resides on the same board as MLDSERV. A more detailed description of the functions of each process is presented below.

1. FIRST STAGE VISION (FSV)

First Stage Vision is the lowest level in the vision processing hierarchy; it acts as the interface between the camera hardware and the upper levels of the vision system. It is responsible for receiving requests for image information and translating those requests to the camera hardware. It is further responsible for reading back run-length data and transferring it to the appropriate common-memory locations for the requesting processes.

FSV controls the commands sent to the Digital Analog Design (DAD) Frame Buffer which reads in an image and converts it to a binary image according to a threshold value supplied by SUP. The binary image is converted to run-length encoded information by DAD and is read back by FSV.

In addition to collecting data for other processes, FSV can also act as a fast ranging mechanism. When appropriately commanded by the vision Supervisor module, it can read in an image and compute the range to the closest point in a given windowed area in real time.

FSV is coded in C and assembly language. The code was developed on a CPM S100 system and is downloaded to an 8086 microprocessor.

2. SECOND STAGE VISION A (SSVA)

SSVA receives commands from Second Stage Vision B (SSVB), and passes them down to FSV. It then waits for FSV to read in an image and pass the run-length encoded data back to SSVA. SSVA performs different operations on the resulting data depending on the kind of picture (flood or structured light) being processed.

On receiving picture data from a floodlit image, SSVA performs a connected-components analysis and constructs a tree of the objects (blobs) in the image. It also computes various mass properties of each blob (area, centroid, and moments) and extracts the boundary. This structure is passed to SSVB.

When the image results from using structured light, SSVA constructs groups of connected curve segments which will be described by SSVB in terms of Chebyshev polynomials. The segments correspond to single curves in the image or to pieces of curves, starting where a curve splits and ending when it ends or when it merges with another curve. The structures are passed up to SSVB, and SSVA polls SSVB until it receives the next command.

The following files contain the code used to implement the actions of SSVA:

ssva.h — This file contains default parameters and constants, as well as descriptions of the structures used to store objects and to pass commands.

ssva.cm — This file, along with vbus.h, are files that define operating system parameters and addresses particular to the individual processor. In general, they are included in other files but are only modified by the operating system.

ainit560.c — This code performs the hardware-dependent initializations. Currently these include setting camera variables and operating system parameters.

ssva.c — This is the main program. It polls SSVB for commands, sends the commands to FSV, and waits for the image to be returned. It then invokes

the connected components procedure or the Chebyshev segmentation and writes the results to SSVB. Error conditions are flagged and reported to the supervisor, and the program goes back to poll SSVB.

conn[12].c — These routines implement the connected-components procedure. They perform the main processing for flood images, constructing the tree structure of blobs in the image and computing properties of each blob. It requires some assembly language subroutines.

interface.c — This program carries out the communications with FSV. It reformats the commands received from SSVB, sends them to FSV, and then waits for the results of the processing performed by FSV.

inicheb.c — This file and the two described below contain the algorithms for segmenting curves for Chebyshev processing. Inicheb.c initializes the line-linking process for structured-light images on a row-by-row basis.

runlen.c — This file contains the algorithm that computes the extent of each curve.

split.c — This program performs some postprocessing to split curves at points of sharp orientation change, i.e. corners.

3. SECOND STAGE VISION B (SSVB)

SSVB receives commands from the first level of the Multilevel Database Processor (MLDSERV) and passes them down to SSVA. It then waits for SSVA to perform connected components analysis or curve segmentation and construct a tree of the objects (blobs) or curves in the image. SSVB accepts the tree of structures and performs feature analysis on each object in the image. For floodlit images, this currently involves finding the corners, principal axis, number of holes, and perimeter of each object. For structured-light images, each segment is described in terms of a polynomial, and the endpoints are reported as features. SSVB sends the tree of components and a set of structures describing the features to MLDSERV and then polls until it receives the next command from MLDSERV.

The following files contain the code used to implement the

action of SSVB:

ssvb.h	This file contains default parameters and constants, as well as descriptions of the structures used to store objects and features and to pass commands.
ssvb.cm	This file and vbus.h are files that define operating system parameters and addresses particular to the individual processor. In general, they are included in other files but only modified by the operating system manager.
init560.c	This code performs the hardware-dependent initializations. Currently these include setting camera variables and operating-system parameters. This file is maintained by the operating system manager. (The same program is used to initialize MLDSERV).
ssvb.c	This is the main program. It polls MLDSERV for commands, sends the commands to SSVA, and waits for the connected components structure or the curve segment structures to be returned. For flood images, it then invokes the various feature extraction procedures for each component and sets up structures describing the results. For structured-light images, Chebyshev polynomials are computed and segment endpoints found. The results are written out to MLDSERV. Error conditions are flagged and reported to SUP, and the program goes back to poll MLDSERV.
readssva.c	This program receives the tree of components or the curve segments from SSVA. It reformats the data into a more useful form for later processing.
cornfast.c	This code finds the corners in the boundary of an object. Cornfast contains a fast algorithm that works on clean boundaries.
cornslow.c	This routine finds the corners in the boundary of an object. Cornslow uses a k-curvature algorithm that takes longer than the algorithm in cornfast.c, but works better. A flag passed by the Supervisor determines which algorithm is called on any particular image.

prinax.c — This file computes the principal axis of an object.

numholes.c — This file finds the number of holes in an object.

perimeter.c — This file computes the perimeter of an object.

perimpma.a86 — This routine is an assembly-language version of the perimeter computation.

ssvbout.c — This routine writes the structures for objects and features or the Chebyshev structures to MLDSERV.

chebfit.c — This routine computes the Chebyshev polynomials for each curve segment.

chebcorn.c — This routine finds the endpoints of Chebyshev segments and builds feature structures to describe them.

4. MULTILEVEL DATABASE AND SERVER (MLDSERV)

The purpose of MLDSERV is to store the results of the lower level processing in a structure appropriate for the upper level recognition algorithms. The server is the portion of MLDSERV which interprets commands received by the Supervisor and attempts to answer the requests. Its main activities are concerned with recognizing objects and extracting relevant information from the database. MLDSERV also performs some of the range and position computations and is responsible for driving the graphic displays.

The following files contain the code used to implement the actions of MLDSERV.

camera.h. — This file contains external declarations of the camera variables.

mldi.h — This file contains default parameters and constants, as well as structure definitions for objects, features and communications with SUP and the lower levels of the vision system.

init560.c — This routine performs hardware dependent initializations. Currently these include setting camera variables used in the range computations and initializing various parameters. (The same

program is used to initialize SSVB).

mldserv.c	This is the main program. It polls SUP for commands to take pictures, for questions to be answered, and for error messages to be displayed. Depending on the input from SUP, either a command is sent down to SSVB to take a picture, the server is invoked to answer a question, or a display routine is called to output an error message. On completion of the task, any errors are reported to the Supervisor, and the polling is resumed.
mldi.c	This routine sends commands to SSVB, calls the routine to set up the entries for the results in the database and causes the 3D position computations to be invoked.
readssvb.c	This program receives the components and features from SSVB. It uses a dynamic storage allocation technique to store the data in a database.
findeqns.c	This routine performs analysis of line-flash images. First, it attempts to pair up line segments. When it succeeds, it uses triangulation to compute the surface equations, orientations, and positions of the implied surfaces. It works with either Chebyshev or "blob" representations of curves.
newcal.c	This routine contains the procedures that implement the camera calibration. The procedures allow transformations from image coordinates to real-world coordinates, and vice-versa (in a number of different coordinate systems).
display.c	This routine contains output routines for printing information about objects and displaying objects and features on a graphics display.
gap.c	This set of routines contains low-level routines for driving the graphics display device.
marlchp.c	This file contains low-level routines for driving the graphics chip.
drawraw.c	This file contains routines for displaying the raw data read by FSV.

caption.c	This routine supplies captions for the displays created by display.c and drawraw.c and the text describing the displays created by mldserv.c and test.c.
malloc.c	This routine contains the dynamic storage allocations and freeing algorithms.
server.c	This is the main program for the question-answering part of the system. It accepts questions and attempts to answer them by looking in the database. Server.c contains the routines for answering questions involving range and orientation information and for answering questions about the largest object in the field of view. It also contains many utility subroutines.
reformat.c	This routine contains procedures to reformat the questions asked by the control system into the structures used internally by the server and to reformat the answers into the structure required by the control system.
fldmatch.c	This routine contains the algorithms used to recognize parts from floodlit images (prismatic parts).
marlch.a86	This is the driver for the Matrox graphics boards 1.
freeall.c	This file contains routines that free dynamically allocated memory for the various structures.
models.c	This file is where the object models are defined and initialized.
quest5.c	This has the routine to begin processing the tray verification request received from MHWS.
quick.c	This contains the algorithm for answering fast flood questions and returning the position of the largest object in the field of view of the camera.
sidelength.c	This routine performs the computations that compare the side lengths of sensed objects with the models and either accepts or rejects

matches on this basis.

5. SUPERVISOR (SUP)

SUP is considered to be the "brains" of the vision system: it accepts commands from either RCS or MHWS, interprets them, translates them into a form suitable for the vision system, and initiates the command sequence in the vision system. If a poor quality image is detected, SUP tries to adjust whatever is necessary to obtain a good picture and then repeats the process. If there are no processing errors, SUP sends the appropriate status and answer to either RCS of MHWS. Simultaneously, SUP checks that each of the vision levels is working. Furthermore, it monitors the SUP keyboard to service any of the interactive features described in Appendix B.

A short description of each of the files is given below:

super.h	This file defines constants and structures used by SUP.
talk.h	This file defines constants and structures used to interface with the control system.
super.cm	This file defines the common memory addresses of modules with which SUP communicates.
sl1.c	This is the main level routine. It contains global variable definitions, initializations, and the main loop. It is responsible for running the vision system and for error checking and compensation.
sl2.c	This section contains utility procedures for the command stack, procedures for getting new commands from the control system, procedures for starting a vision command, and procedures for sending answers back to RCS or MHWS.
sl3.c	This procedure reads and processes errors from FSV and SSVA.
sl4.c	This procedure reads and processes errors from SSVB and MLDSERV.
sl5.c	This routine contains the procedures which communicate with RCS through the 589 card.

s16.c. This file contains procedures for user interaction (operator service).

s17.c This file contains procedures for monitoring the vision command service.

defaultt.c This routine sets up question 5 for a default tray verification request.

makeparm.c This file sets up camera calibration parameters for the camera configuration in the AMRF.

netport.a86 This code interprets signals emitted by a remotely operated push button for generating an interactive tray verification request.

6. NETBOARD (NET)

The NET process acts as an interface between the vision system (SUP in particular), MHWS and the AMRF database. All functions contained in NET are activated by commands from SUP. These functions include database initialization requests, checking for MHWS requests, and issuing requests for tray definition reports and tray contents reports. Communication between NET and either the AMRF database or MHWS is done over the AMRF network. A brief description of the routines resident on the NET board follows:

netbd.h This is an "include" file containing definitions of structures and constants used by NET and SUP.

commun.h This is an "include" file describing the structure of command requests and status expected by MHWS as well as the structure of mailbox communications.

address.h This file declares physical addresses used by NET in communicating with the AMRF database and MHWS.

net.cm This file contains vision system definitions of physical addresses used for inter process communications.

mbus.h This "include" file contains the definitions of structures and constants used for maintaining inter process communications.

netmain.c This is the main loop of the NET process. It

polls on requests from SUP and then calls the appropriate routines to carry out the requests. Status and answers are returned to SUP at the completion of each command.

initnet.c — This routine contains routines for starting the AMRF database communications: ABORT, INITIATE, and STARTUP.

netcommand.c — This code polls on tray verification commands from MHWS. When one is received, it is read and the appropriate status is returned to SUP.

writenet.c — This file contains routines for generating status reports to MHWS upon completion of a tray verification sequence.

dbio.c — This routine constructs the status message to be returned to MHWS.

newmail.c — This file contains routines for reading and writing network mailbox communications.

netio.c — This file is responsible for interprocess communication between SUP and NET.

The following routines are responsible for formatting or unpacking AMRF database communications: asnoi.c, userdsea.c. These routines use definitions provided by asn.h, asnapp.h, basic.h, imdassts.h, and userdse.h.

7. PHYSICAL DESCRIPTION OF THE VISION SYSTEM:

A General Electric Model 2500 camera is mounted on the T3 robot wrist in HWS, and a second camera is mounted on the gantry at the tray verification station. A flash box is attached to the robot mounted camera: this can be software activated to emit either a double planes of light flash or a point light flash. Double planes of light are used for determining range, pitch, and yaw of the object in view. The point light flash is used to compute the azimuth and roll of the object.

The image received from the camera is captured by a Digital Analog Design (DAD) Frame Buffer. The DAD hardware provides the ability to capture an image and threshold it in accordance with software provided values. The result of the thresholding operation is a binary image in which all grey level values greater than the supplied threshold are converted to white, while

those values below the threshold are converted to black. The binary image is then compacted into run-length encoded data, i.e., only transitions from black to white or white to black are recorded. In addition to generating run-length encoded data, the DAD hardware also can filter "noisy" pixels from an image upon software command or can read and transfer a full grey scale image.

The vision system rack which is resident in the AMRF contains 8086 microprocessor boards for the vision processes as well as microprocessors for communication purposes. MLDSERV is executed from an S100 board also located in the vision rack.

IV. SYSTEM OPERATION

To start the vision system, locate the vision system rack which is to the right of the vision work area on the AMRF floor (see Figures 1 and 2). After opening the rear door of the rack, turn on the power strip to the left. Also, turn on the power strip on the right side of the three vision monitors that are in front of the glass window in the hallway. Plug in the lamp that is attached to the gantry at the tray verification station which is located in front of the Turning Workstation.

At the vision work area, turn on the power strip to the right of the system monitors. To boot the S100 system, hit the carriage return key <CR> twice on the center keyboard. After the system has completed booting, the "H>" prompt will appear on the screen. If a system message followed by the system prompt does not appear within 10 seconds, toggle the switch labelled "S100 Reset" on the switch box up and down. Then hit the carriage return key twice on the S100 monitor and wait for the "H>" prompt. Type "USER 10<CR>" on this same keyboard. Next, type "DOWNPARA -A ALLRL -G<CR>". The S100 monitor will display comments about which process code it is downloading. The far right monitor, the Supervisor monitor, will display information concerning the status of the vision system start up and which processes are successfully running.

Two messages should appear next: one on the S100 monitor asking whether to enter in tray corner coordinates, and one on the Supervisor monitor asking for a <CR> when the network connections are present. The operator response to the tray corner coordinate query should always be "0" followed by <CR>. (Tray coordinates provide for the alignment of the lower left corner of the tray with the lower left corner of the image. They should be entered manually only if the tray position on the cart has changed significantly from past usage.) The prompt on the Supervisor requires only a "<CR>" whenever the operator has verified that the vision system's network connections to MHWS and the database have been established. At the completion of the downloading procedure, the message "THE VISION SYSTEM IS UP!" will appear on the Supervisor monitor. No further response is required of the operator. The vision system will run unattended until it is powered off.

Any variation on this sequence of events implies that the vision system is not running properly. The easiest remedy is to restart and redownload the vision processes. This can be done using the switchbox to the right of the vision system process monitors. Flip the lever marked "global" on the face of the box

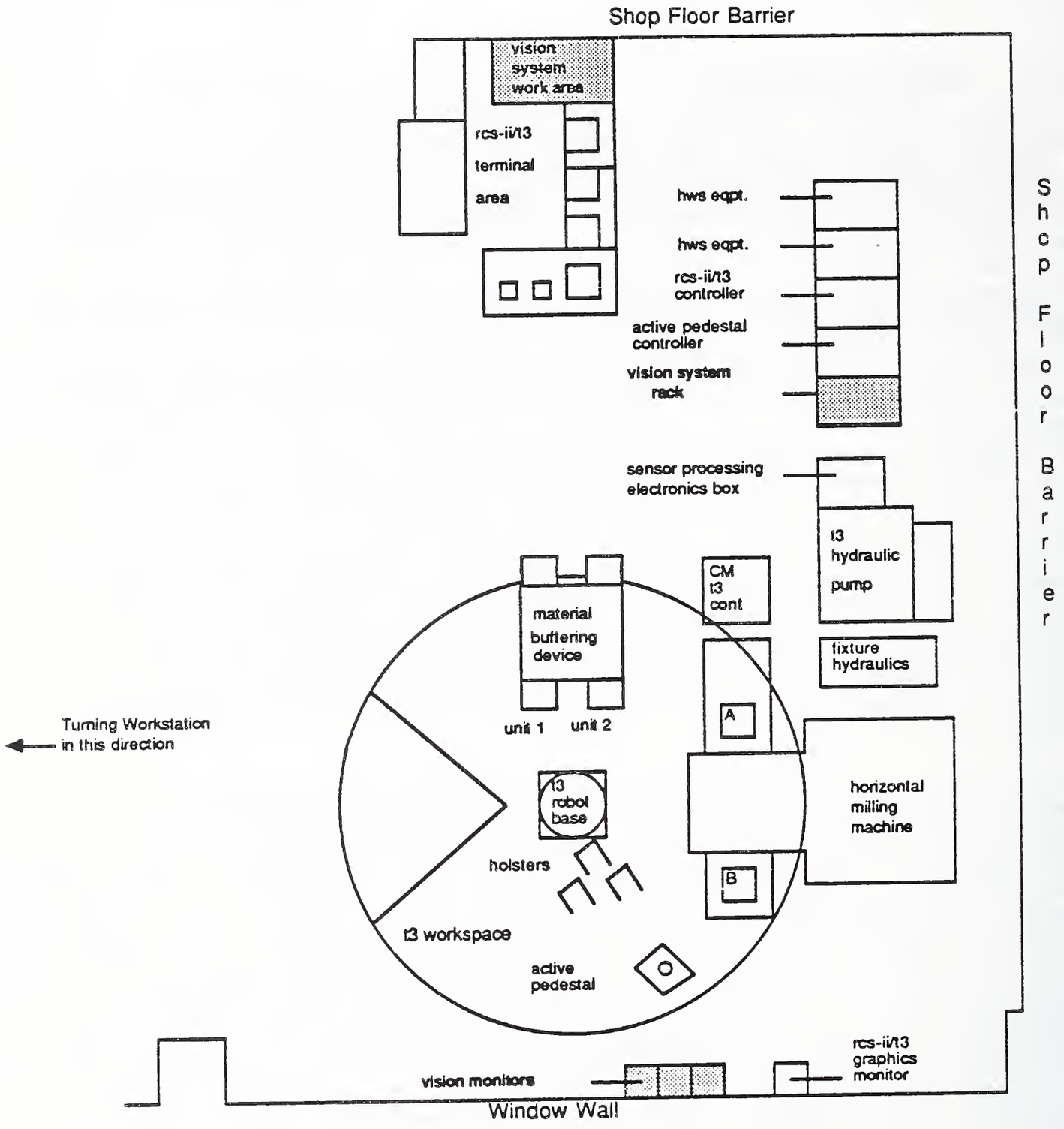

Figure 1. AMRF Shop Floor

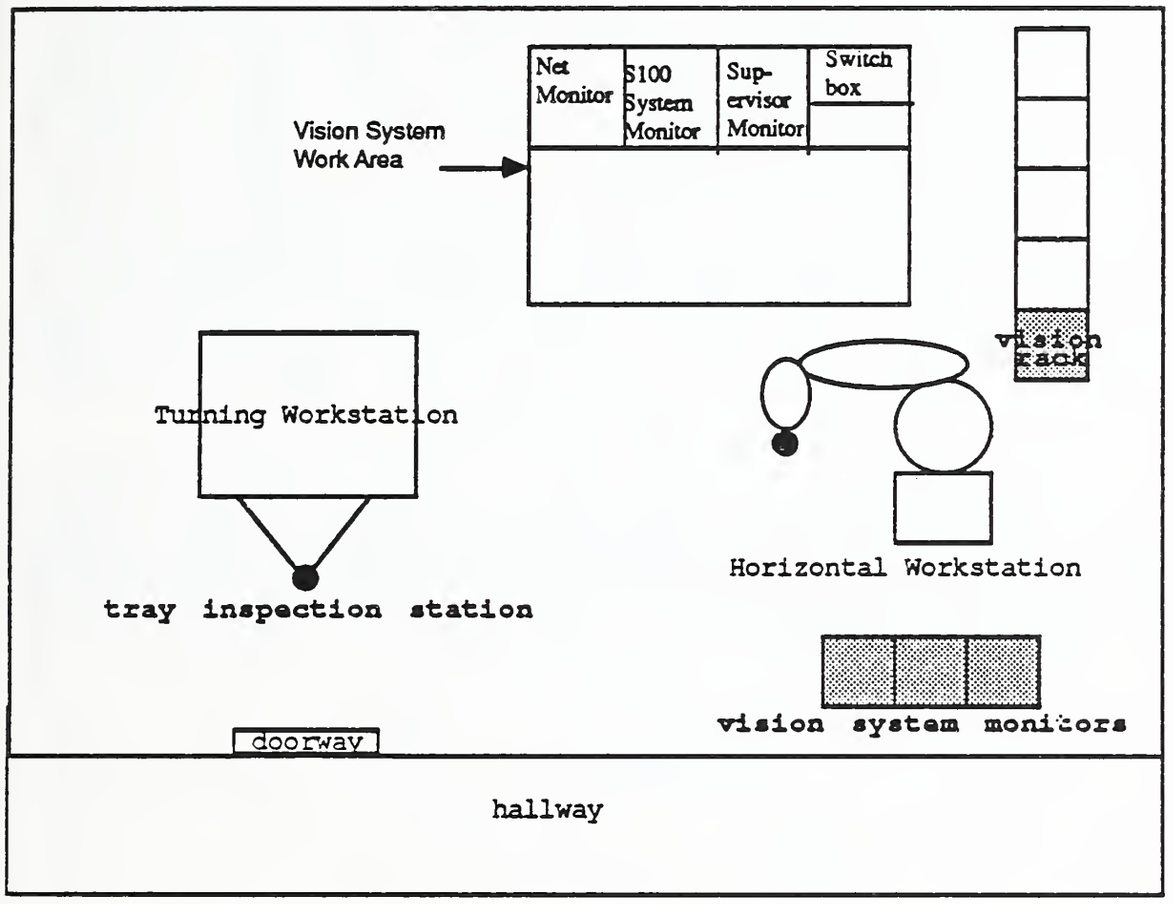

Figure 2. Enlargement of Vision System Work Area

to the up position, then press the button marked "reset". Then restart the system by reentering "DOWNPARA -A ALLRL -G" on the S100 keyboard.

The vision system has two modes of operation. During normal command mode, the vision system will poll and wait on commands from either RCS or MHWS (see Section II). In simulated command mode, the operator can enter a command from the keyboard (see Appendix B). If the RCS is present and sends command "0", the vision system will generate its own simulated commands. The monitor on the left displays the status of the network process, NET, and will show the interaction of the vision system with MHWS. There are two other monitors that are used for displaying results of vision recognition tasks. The monitor on top of the SUP monitor displays the field of view that the current camera sees. The monitor on top of the S100 monitor is a graphic representation of the results of the most recent vision task.

To shut down the vision system, turn off the three power strips that are referred to in the initial power-up sequence and unplug the light above the tray inspection camera. Close the doors on the vision system rack securely.

V. SYSTEM INTERFACE

The vision system processes communicate via a common memory area governed by a file system. The system operates in both a top-down and bottom-up manner (see Figure 2). The multiprocessing levels of the vision system pass raw data using a bottom-up process. FSV is responsible for commanding the camera to take a picture. The image data is passed to SSVA for processing and then to SSVB for further analysis. The multiprocessing levels perform object recognition in a top-down manner. SUP is responsible for starting and checking all levels of the vision system and for accepting commands from RCS. NET accepts commands from the Material Handling Work Station (MHWS). MLDSERV reformats these questions so that they may be passed down to the other processes.

The interprocess communications are handled using the vision system's MBUS library. The purpose of the MBUS library is to ensure data integrity of communications and to allow for asynchronous communications between processes. Blocks of predefined common memory, called "files", accessible to each vision process are defined. A set of flags associated with each file records its current state (i.e, which process owns the file, is it currently open or closed, which process accessed it last, and the system time of last access). Only those processes granted read and/or write permission can access a file. A user can only access a file if it is closed and must open the file before it can be either read or written. The concept of opening and closing files is analogous to the UNIX file system calls.

Communications betewen the RCS and the vision system are handled using a high speed parallel link. A master-slave relationship is represented on the board to describe the relationship between the two systems and to indicate the direction of communication. Communication is initialized by the RCS setting the appropriate bits. Communication protocol and a command structure are used to allow the RCS to request a command from the vision system (see Appendix A).

The MHWS and the vision system communicate using the VAX common memory mailbox system. Mailbox areas are established when the network communications are made to provide for message passing between the two systems and between the vision system and the AMRF database. When first brought up, the vision system initializes with the database by sending the UVA protocol commands, "ABORT", "INITIATE", and "STARTUP" [1]. Vision then polls the mailbox command area for a change in sequence number to detect when a tray verification request has been sent from MHWS.

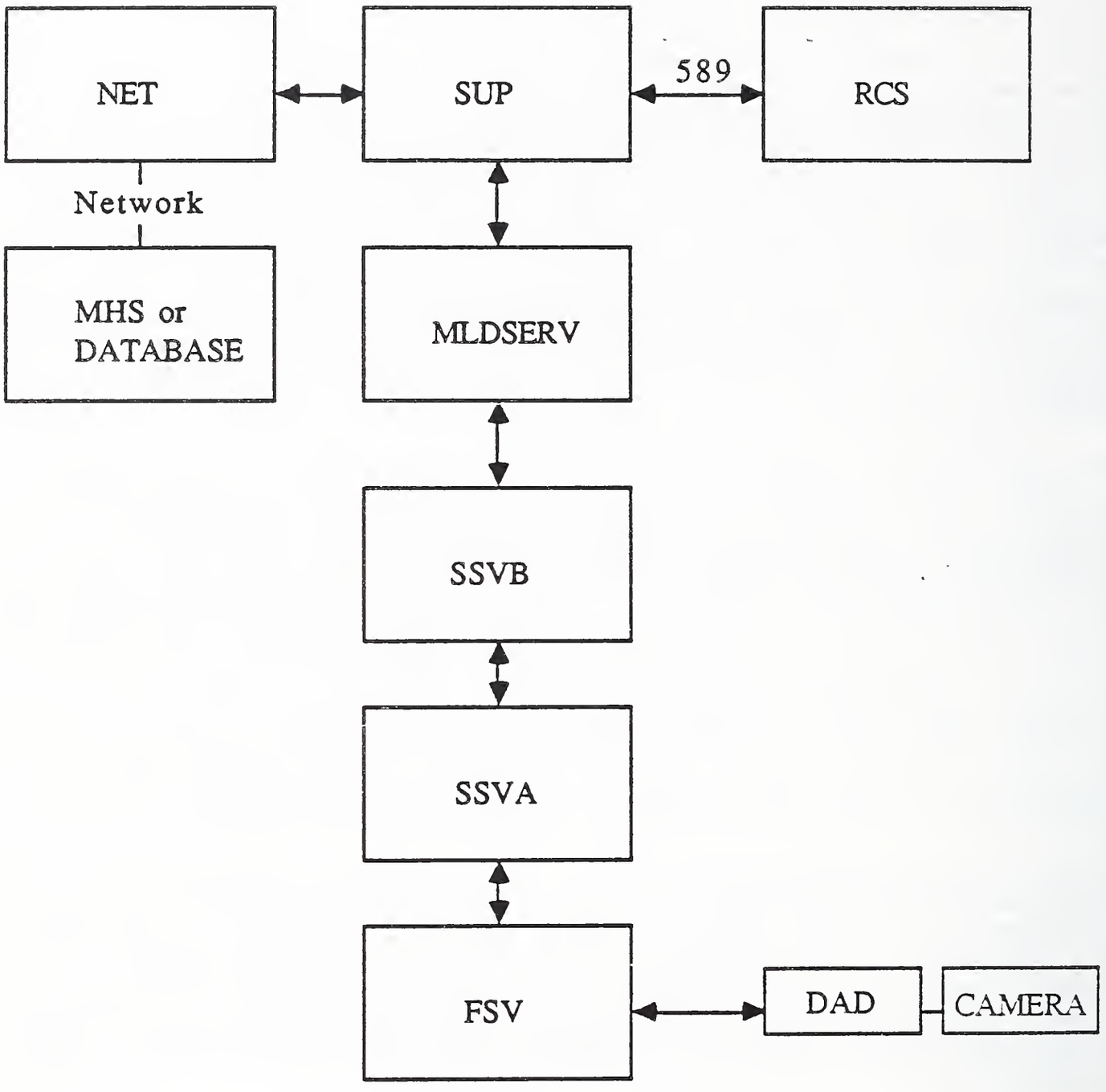

Figure 3. NBS Vision System

A question which describes a default tray configuration is then posed to the vision system. Upon completion of the recognition task, a status report is sent to MHWS in the status mailbox.

VI. FUTURE PLANS

Future plans include expanding the system software to make it more versatile. The MHWS interface will be able to send UVA protocol commands and to interpret more sophisticated status messages. MHWS will be able to send commands to establish the configuration between itself and the vision system. Vision will also utilize its capability to notify MHWS of the status of the verfication request and to detail either reasons for failure or to update part locations.

The database interface will be used to extract information relative to any particular tray and to store updated information produced by the vision system. An MHWS command will be decomposed to obtain the necessary keys for vision to access a tray contents report and a tray definition report from the database. Information from these data reports will be used to construct a tray verification question. The updated part positions will be posted to the database.

REFERENCES

[1] Nanzetta, P. and Rippey, W., "Integration of the AMRF", [To Come].

APPENDIX A. DATA STRUCTURES

1. RCS COMMUNICATIONS

Following are the input and output structures defined between RCS and SUP:

```
# define CMDLNGTH 500              /*  length of the command data
                                       field  */

# define RSPSLENGTH 500            /*  length of the response
                                      data field  */

struct inbuffer {

          int bytecount;           /* length of question */
          int qtype;               /* question number    */
          int cyclecount;          /* count of control system -
                                      incremental request
                                      number */
          int writetime;           /* real time when the question
                                      was written */
          int iresrvi;             /* reserved locations--future
                                      expansion */
          int in_error;            /* control system error messages
                                      -- currently unused*/
          int ireserv2;            /* reserved location--future
                                      expansion */
          int ireserv3;            /* reserved locaton--future
                                      expansion */
          char data[CMDLENGTH];    /* parameters for the question*/

          };

struct outbuffer {
          int bytecount;           /* length of answer */
          int qtype;               /* question type -- same as
                                      for the question */
          int cyclecount;          /* same as for the question */
          int anstime;             /* real time that answer
                                      buffer written */
          int questime;            /* real time that question
                                      buffer written  */
          int out_error;           /* errors passed to the
                                      control system */
          int oresrvi;             /* reserved location--
                                      temporarily used for
                                      pictries */
```

```
        int oresrv2;            /* reserved location */
        char data[RSPSELENGTH];/* data for the answer */
        };
struct prog589  {               /*  Description of bytes
                                    used for communication
                                    between RCS and vision  */
        char cmdbyte;
        char statusbyte;
        char chp1;
        char chp2;
        char chp3;
        char chp4;
        char devnum1;
        char dtype1;
        char mempt1;
        char mempt2;
        char mempt3;
        char mempt4;
        char devnum2;
        char dtyppe2;
        char mempt5;
        char mempt6;
        char mempt7;
        char mempt8;
        char bcount1;
        char bcount2;
        char bcount3;
};
```

2. NETWORK COMMUNICATIONS

2.1 Mailbox Transfers

```
struct mgrm {                   /* mailgram structure */
     int mg_seq;                /* sequence number */
     int mg_len;                /* mailgram length */
     char mg_txt[MAXMGM];       /* text of mailgram */
};

struct mbox {                   /* mailbox structure */
     int mb_wlock;              /* write lock semaphore */
     int mb_rlock;              /* read lock semaphore */
     struct mgrm mbmgrm;        /* mailgram */
};
```

2.2 Database Transactions

```
struct work_element_def {
```

```
        char wder_id[16];
        char wupdate_number[16];
        char waction[16];
        char wworkelem[16];
        char wnr_parms[16];
        char watt1[16];
        char wval[16];
        char watt2[16];
        char wval2[16];
        char watt3[16];
        char wval3[16];
        char watt4[16];
        char wval4[16];
};

struct part_def {
        char psector [16];
        char pitem_serial_nr [16];
        char pitem_name [16];
};
struc value_def {
        char name [32];
        char type [16];
        char width [16];
};

struct performance_report_def {
        char prreport_name [32];
        char prsystem_id [16];
        char prwork_element [16];
        char prnr_values [16];
        struct value_def prvalue [108];
};

struct order_performance_report {
        char opreport_name [32];
        char opsystem_id [16];
        char oporder_id [16];
        char oppfr_version [16];
        char opsupervisor [16];
        char opsactual_start [32];
        char opcactual_completion [32];
        char opduration [32];
        char opnr_sub_orders [16];
        char opnr_parameters [16];
        char opsub_order [16];
        struct work_element_def opwork_element;
        struct performance_report_def opperformance_report; };
```

```
struct date_time {
     char dreport_name [32];
     char dcurrent_time [32];
};

struct sector_def {
     char ssector [16];
     char sx_offset [8];
     char sy_offset [8];
     char sz_offset [8];
     char sx_dimension [8];
     char sy-dimension [8];
     char sz_dimension [8];
};

struct tray_definition_report {
     char tdreport_name [32];
     char tditem_subtype [32];
     char tdnr_locations [16];
     struct sector_def tdsector [9];
};

struct tray_contents_report {
     char tcreport_name [32];
     char tctray_id [16];
     char tcitm_subtype [16];
     char tctray_clear [16];
     char tcnr_locations [16];
     struct part_def tcpartpos[9];
};

typedef struct _Transaction{
     ByteType itype_transaction;
     char itransaction_id[4];
     char old_user_id[8];
     char old_transaction_id[4];
     char *dml_string;
     struct _Transaction *nnext;
}_Transaction;

typedef struct {
        struct {
                SeqType ncommand_number;
                char userid [8];
        } fixed_segment;
        -Transaction *dtransaction;
} DataServerCommand;
```

```
struct TransactionStatus {
     char stransaction_id[4];
     ByteType stype_transaction;
     struct    {
               ByteType summary_status;
               ByteType padbt;
               long int detail_status;
     }status_elements;

     IS0646String error_message;     /* opt., 0 or "" if none */
     char data_reference[4];         /* opt., "" if none */
     int byte_count;                 /* opt., < 0 if none */
     int row_count;                  /* opt., < 0 if none */
     char source_station[8];         /* opt., "" if none */
     /* private */
     struct TransactionStatus *next;/* 0 if no more */
};

struct _DataServerStatus{
        struct {
               char s_userid[8];
               char serverid[8];
               TimeType report time;
               SeqType report_number;
               SeqType nlast_command_number;
               CharType server_status;
               CharType user_link_status;
               CharType master_link_status;
               ByteType pad_byte;
        } ds_fixed_segment;
        TransactionStatus *transaction_status;
};
```

2.3 MHWS Transactions

```
struct com_message {
     char cmsg_id [4];
     int ccom_len;
     int ccom_no;
     char ccom_time [12];
     int cxact_len;
     int cder_len;
     int crce_len;
     int ctrns_actup;
     char ctrns_keywd [16];
     int ctrns_cntparam;
     char ctrns_params [16];
     int cder_num;
```

```
    struct work_element_def cwork_element;
};

struct status_message {
    char smessage_id [4];
    int slen;
    int snum;
    char stime [12];
    int secho_len;
    int strns_len;
    int sder_len;
    int srce_len;
    int scomno;
    char scomtime [12];
    int strns_statup;
    char strns_keywd [16];
    int strns_paramcnt;
    int sder_num;
    struct work_element_def swork_element;
};
```

3. COMMUNICATIONS BETWEEN MLDSERV AND SUP

3.1 Data Structures for the Questions:

```
struct q1 {
        unsigned int q1objid;       /* dummy entry for
                                       compatibility of
                                       questions*/
          int q1range;              /* expected range to object */
};

struct q2 {
        unsigned int q2objid;       /* expected object name */
        int q2range;                /* expected range to object*/
};

struct q3 {
        unsigned int q3objid;       /* expected object name */
        int q3range;                /* expected range to object */
        int q3cosx;                 /* directed cosines of normal
                                       to surface */
        int q3cosy;
        int q3cosz;
};

struct q4 {
       int q4objid;
```

```
};

struct q5 {
        int sector;                     /* sector number */
        float origin[2];                /* x,y position of sector
                                           origin */
        float size[2];                  /* x,y position of sector
                                           size */
        char identifier[16];            /* part identifier */
        int flag;                       /* part verified?
                                           'no' =0 'yes' or 'no part'
                                                =1  */
        unsigned int q5error;           /* reason for failure */
        float qx, qy, qz;               /* position of part centroid
                                           */
        float qorientation;
};
```

3.2 Structures for the Answers:

```
struct a1 {
        int aldummy[3];                 /* dummy entry for answer
                                           compatability */
        int alcosx;                     /* cos of x component of angle
                                           between lens axis and line
                                           to centroid of blob */
        int alcosy;                     /* cos of y component of angle
                                           between lens axis and line
                                           to  centroid of blob  */
        int x_blob_centroid;            /* observed x coordinate of
                                           centroid if height was
                                           given, otherwise 0 */
        int y_blob_centroid;            /* observed y coordinate of
                                           centroid if height was
                                           given, otherwise 0 */
};

struct a2 {
        unsigned int a2objid;           /*  expected object name */
        int confidence;                 /*  confidence in
                                            identification (computed
                                            as difference between
                                            between  expected and
                                            computed distances */
        int best_obj_fit;               /*  actual best id */
        int z_axis_range;               /*  observed range to object
                                             z axis */
        int roll_angle;                 /*  observed roll angle of
```

```
                                        object */
        int x_pos_centroid;             /*  observed x coordinate of
                                            centroid*/
        int y_pos_centroid;             /*  observed y coordinate of
                                            centroid*/
};

struct a3 {
        unsigned int a3objid;           /*  expected object name */
        int a3dummy[2];                 /*  compatibility with other
                                            answers*/
        int obs_range;                  /*  computed range to object
                                            */
        int cosx_obs;                   /*  directed cosines normal
                                            to surface */
        int cosy_obs;
        int cosz_obs;
 };

struct a4 {
        float led[4][2];                /*  centroids of 4 leds*/
};

struct a5 {
      int sector;                       /*  sector number */
      float origin[2];                  /*  x,y position of sector
                                            origin */
      float size[2];                    /*  part identifier */
      int flag;                         /*  part verified?
                                            'no' = 0   'yes' or 'no
                                             part'= 1 */
      unsigned int q5error;             /*  reason for failure */
      float qx,qy,qz;
      float qorientation;
};

struct model {                          /*  models used in server */
      char *modname;                    /*  ASCII name */
      float modarea;                    /*  area of object in square
                                            mm */
      float modperim;                   /*  perimeter in mm */
      float modaxis;                    /*  principal axis
                                            calculated */
      float longside;                   /*  length of long side - in
                                            mm */
      float shortside;                  /*  length of short side */
      float modheight;
      float bounds[6];                  /*  bounds on range of
```

```
                                        measured values */
                                /*  bounds[0] is distance slop
                                    for flood */
                                /*  bounds[1] is distance slop
                                    for line */
                                /*  bounds[2] is area slop */
                                /*  bounds[3] is side length
                                    slop */
                                /*  bounds[4] is side ratio
                                    slop */
                                /*  bounds[5] is radius slop
                                    */
        int modholes;           /*  number of holes */
};
```

3.3 Model Data for the Objects

```
struct model proto[]= {
/*  box bottom */          "box bottom",   7137.0,     340.0,
                                              0.0,      95.25,
                                             74.93,     36.83,
                                              1.1,       1.1,
                                              1.1,       1.06,
                                              0.1,       1.1,
                                              0,
/* box top */              "box top ",     7137.0,     340.0,
                                              0.0,      95.25,
                                             74.93,     17.78,
                                              1.1,       1.1,
                                              1.1,       1.06,
                                              0.1,       1.1,
                                              0,
/* flag */                 "flag",         7679.0,     350.5,
                                              0.0,      87.63
                                             24.13,      1.1,
                                              1.1,       1.15,
                                              1.08,      0.1,
                                              1.1,         0,
/* cylinder */             "cylinder",      469.97,     158.0,
                                              0.0,      50.3,
                                             50.3,      25.4,
                                              1.1,       1.1,
                                              1.25,      1.08,
                                              0.1,       1.1,
                                              1,
/* hole in cylinder */     "hole",         1516.5,     138.05,
                                              0.0,      43.9,
                                             43.9,      25.4,
```

```
                                                    1.1,        1.1,
                                                    1.15,       1.08,
                                                    0.1,        1.1,
                                                    0,
/* dog                          "dogold",          3225.8,     266.75,
                                                    0.0,        101.6,
                                                    31.75,      31.75,
                                                    1.1,        1.1,
                                                    1.15,       1.2,
                                                    0.22,       1.1,
                                                    0,                */
/* ring*/                       "ring",            235.7,      62.86,
                                                    0.0,        0.0,
                                                    0.0,        0.0,
                                                    1.1,        1.1,
                                                    1.2,        0.0,
                                                    0.0,        0.0,
                                                    0,
/* turning adapter */           "thold",           5741.924,   307.848,
                                                    0.0,        90.424,
                                                    63.5,       63.5,
                                                    38.1,       1.1,
                                                    1.1,        1.1,
                                                    1.09,       0.08,
                                                    1.1,
 /* boring adapter */           "bhold",           5741.924    307.848,
                                                    0.0,        90.424,
                                                    63.5,       50.8,
                                                    1.1,        1.1,
                                                    1.1,        1.09,
                                                    0.08,       1.1,
                                                    0,
/*  pipe flange 205 */          "f1205-b",         2199.092,   195.58,
                                                    0.0,        62.738,
                                                    35.1,       23.0,
                                                    1.1,        1.1,
                                                    1.1,        1.1,
                                                    1.09,       0.08,
                                                    1.1,
/* pipe flange 207 */           "f1207-b",         4143.167,   266.71,
                                                    0.0,        84.074,
                                                    49.28,      27.0,
                                                    1.1,        1.1,
                                                    1.1,        1.09,
                                                    0.08,       1.1,
                                                    0,
/* pipe flange 209              "f1209-b",         7646.760,   358.65,
                                                    0.0,        109.47,
```

```
                                                    69.85,      42.9,
                                                     1.1,        1.1,
                                                     1.1,        1.09,
                                                     0.08,       1.1,
                                                     0,
/* plunger bracket */       "brack-b",          1862.667,    252.77,
                                                     0.0,        72.974,
                                                    34.93,      40.46,
                                                     1.1,        1.1,
                                                     1.1,        1.2,
                                                     1.1,        0.08,
                                                     1.1,
/* IGES test part */        "iges-b",           6541.77,     360.08,
                                                     0.0,        129.54,
                                                    50.5,       50.5,
                                                     1.1,        1.1,
                                                     1.1,        1.09,
                                                     0.08,       1.1,
                                                     0,
/* valve body */            "valvem-b",        18749.59,     854.61,
                                                     0.0,        222.50,
                                                   114.3,      120.65,
                                                     1.1,        1.1,
                                                     1.1,        1.1,
                                                     0.08,       1.1,
                                                     0,
/* pen set base            "penset",          13200.0,      502.0,
                                                     0.0,        176.0,
                                                    75.0,       18.0,
                                                     1.1,        1.1,
                                                     1.1,        1.06,
                                                     0.1,        1.1,
                                                     0,          */
/* 14:  locking clevis */"lockg clevis",5903.21      312.93,
                                                     0.0,        92.964,
                                                    63.5,       50.8,
                                                     1.1,        1.1,
                                                     1.1,        1.06,
                                                     0.1,        1.2,
                                                     0,
/* 15:  hinge block */      "hinge block",      3654.0,      242.0,
                                                     0.0,        63.0,
                                                    58.0,       32.0,
                                                     1.1,        1.1,
                                                     1.1,        1.05,
                                                     0.1,        1.2,
                                                     0,
/* 16:  link head */        "link head",        1575.0,      163.0,
```

```
                                                    0.0,      50.0,
                                                   31.50,     50.50,
                                                    1.2,       1.1,
                                                    1.3,       1.06,
                                                    0.25,      1.1,
                                                    0,
/* 17:  link bar */          "link bar",          1616.0,     234.0,
                                                    0.0,     101.0,
                                                   16.0,      16.0,
                                                    1.2,       1.1,
                                                    1.25,      1.20,
                                                    0.25,      1.1,
                                                    0,
/* 18:  block */             "block",             6750.0,      37.00,
                                                    0.0,     135.0,
                                                   50.0,      50.0,
                                                    1.1,       1.1,
                                                    1.1,       1.06,
                                                    0.0,       1.1,
                                                    0,
/* 19:  dog */               "dog",               4066.0,     290.0,
                                                    0.0,     107.0,
                                                   38.0,      38.0,
                                                    1.2,       1.1,
                                                    1.3,       1.06,
                                                    0.25,      1.1,
                                                    0,
/* null model to end */      "",   0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                                   0.0, 0.0, 0.0, 0.0, 0.0, 0,

};
```

4. FILE COMMUNICATIONS

```
struct filebuf {                    /*  one structure for each
                                        file*/
     char       filename[16];
     char       status;             /*  open or closed (this flag is
                                        internal to program) */
     uchar      prevuser;           /*  file is opened */
     uchar      otheruser;          /*  userid of other process
                                        using this file */
     char       owner               /*  =1 if this processor "owns"
                                        file */
     unit       otherusers;         /*  Each bit is a user.  Bit 15 set
                                        means this is the owner */
     uchar      multiple;           /*  True says more than one reader
                                        at a time may access file */
```

```
    uchar     uninitflags;   /*  Bit 0 set says we've already
                                 printed a msg   */
                             /*  Bit 1 set says flag is
                                 unitialized (illegal)   */
    struct                   /*  (flagaddr currently resides in
                                 segment x2860)  */
    flagbuf   *flagaddr;     /*  upper byte=last user, lower
                                 byte = open/closed */
    unit      segment;       /*  segment in which buffadr,
                                 endaddr, next lie */
    char      *buffaddr;     /*  starting address of file */
    char      *endaddr;      /*  last valid address plus one */
    char      *next;         /*  address of next read or write
                                 position  */
};

struct flagbuf {             /*  this lies in FLAGSEG*/
                             /*  (hence we don't reserve
                                 storage for it here)  */
    char      openflg;
    uchar     curruser;

    char      dummyAA;       /*  used by elimFF to distinguish
                                 bus timeouts  */
                             /*  dummyAA must NEVER be set to FF
                                 by any process  */
    char      dummyxx;

    unit      opentime;      /*  time file was last opened  */
    unit      closetime;     /*  time file was last closed  */
    unit      rdusers;       /*  bit vector of users
                                 currently reading file  */
                             /*  (for multi-reader files
                                 only) */
    unit      writereq;      /*  bit vector of user (if
                                 any) wanting (or */
                             /*  having) write permission.
                                 */
    unit      futurespace;
    unit      peektime;      /*  time flag was last peeked
                                 at */
};

struct    allfilebuf  {      /*  one structure for
                                 each file  */
    char      a_filename[16];
    unit      a_uservec;     /*  bit id's of other  process
                                 using this file  */
```

```
    unit        a_owner;        /* bit id of "owner" of file    */
                                /* (flagaddr currently
                                   resides in x2860)  */

    struct flagbuf *a_flagaddr;
                                /* address of system flagbuf for
                                   this file */
    unit        a_segment;      /* segment in which buffadr,
                                    endaddr, next lie  */
    char        *a_buffaddr;    /* starting address of file   */
    char        *a_endaddr;     /* last valid address plus
                                   one  */
    int         a_multiple;     /* True means more than one
                                   reader at a time may
                                   access file */
    int         a_dummy2;       /* make allfilebuf an even 32
                                   bytes  */
};
```

5. OBJECT DESCRIPTION

```
struct ssvaobj {                /* the structure received
                                   from ssva  */
     long8 aarea,axcntr;
     long8 aycntr,am20,am11;
     long8 di, d2;              /* scratchpad for ssva */
#ifdef MOMENTS
     long8 amm10, amm01, amm20,
     amm02, amm11, amm21, amm12,
     amm03, amm30;
#endif
     int ast_list;              /* pointer into edge-list */
     int aparent, achild,
     asibling, acolor, aximin,
     axmax, aymin,aymax;
     int comp;                  /* component number */
     int ah_area, al_area;
     struct frame *aframe;      /* pointer to header structure
                                   for picture  */
     int awhole;                /* 1 if the blob was
                                   completely processed, 0
                                   otherwise  */
};

struct objects {
     double area;               /* area of the component */
 double xcntr, ycntr;           /*  the centroid of the object
                                    */
```

```
double m20, m11;                /*  object moments  */
double mm10, mm01, mm20,
mm02, mm11, mm21, mm03, mm30;
float perim;                    /*  index to length of
                                    boundary (not including
                                    holes) */
unsigned int equation;          /*  index of structure
                                    containing surface
                                    equation */
int holes;                      /*  number of holes  */
unsigned int olinks[2];         /*  links to other objects  */
int future;                     /*  link to next frame  */
int past;                       /*  link to previous frame  */
int odescription;               /*  further information about
                                    the object  */
int ox2d;                       /*  2d position: x, y */
int oy2d;
float oaxis2d;                  /*  2d principal axis
                                    (radians)  */
int ox3d;                       /*  3d position: x, y, z */
int oy3d;
int oz3d;
int oyaw;                       /*  3d position:  yaw, pitch,
                                    roll (degrees)  */
int opitch;
int oroll;
unsigned int st_list;           /*  start address of edge
                                    list  */
unsigned int s_corn, e_corn:    /*  start and end
                                    pointers for corners  */
unsigned int parent, child,
sibling;                        /*  links in tree of
                                    components  */
int color;                      /*  object or hole  */
int xmin, xmax, ymin, ymax;     /*  bounding rectangle  */
unsigned int h_area, l_area;    /*  links to next largest
                                    and next smallest
                                    components  */
int ocoordsys;                  /*  which coordinate system is
                                    used for position  */
struct frame *oframe;           /*  pointer to header
                                    structure for picture  */
int otype;                      /*  object type or number */
int oname;                      /*  name from model database*/
int opart_of;                   /*  which object it belongs to
                                                        */
```

```
    int oconfidence;              /*  is this really the right
                                      object?  */
    int whole;                    /*  1 if completely processed
};                                    by ssva, 0 otherwise  */
```

5.1 Structures for Features in Database:

```
struct feature  {
    double surfeqn[4];            /* surface equation if
                                      feature is a surface  */
    int  fx2d;                    /*  2d position:  x,  y  */
    int  fy2d;
    float  faxis2d;               /*  2d angle (radians)  */
    int  fx3d;                    /*  3d position:  x,  y,  z,
                                      yaw,  pitch,  roll  */
    int  fy3d;
    int  fz3d;
    int  fyaw;                    /*  3d position:  yaw,  pitch,
                                      roll  (degrees)  */
    int fpitch;
    int froll;
    unsigned int flinks[2];       /*  links to other features
                                      */
    unsigned int fdescription;    /*  further information about
                                      the feature  */
    int fedgenum;                 /*  number of edge points if
                                      edge feature  */
    unsigned int fcomponent[3];   /*  which components(s) it
                                      belongs to  */
                                  /*  e.g., flood blobs  */
    int fwindow[4];               /*  coords of window it was
                                      detected in  */
    int ftype;                    /*  feature type or number  */
    int fname;                    /*  name from model database
                                      */
    struct frame  *fframe;        /*  pointer to header
                                      structure for picture  */
    unsigned int fpart_of;        /*  which object it belongs to
                                      in ssvb  */
    int fconfidence;              /*  is this really the right
                                      feature?  */
    int fcoordsys;                /*  which coordinate system is
                                      used for position  */
};

struct edges  {
    int l_link,x_coord,y_coord,r_link;
};
```

```
struct cheby  {                          /*  The Chebyshev
                                             coefficients, errors, and
                                             line endpoints  */
     int firstx;                         /*  image coordinate start and
                                             end points of segment  */
     int lastx;
     double coeffa;                      /*  coefficients of polynomial
                                             */
     double coeffb;                      /*  origin is at firstx  */
     double coeffc;
     double coeffbb:                     /*  (alternate coefficients)
                                             */
     double coeffcc;                     /*  origin is at midpoint of
                                             curve  */
     double cerror;                      /*  fitting error  */
     unsigned int *cequation;            /*  equation of surface  */
     unsigned int cnumfeats;             /*  number of features  */
     int chebfeats;                      /*  pointer to corners of
                                             segment  */
     int nextcheb;                       /*  next structure of this
                                             type  */
     struct frame  *cpicture;            /*  pointer to frame info  */
};

/*  Chebyshev structures  */

struct vert_seq  {
                int a;                   /*  dark to light
                                             transition */
                int b:                   /*  light to dark
                                             transition  */
                int center;              /*  center = (a + b)/2 */
                int link;                /*  pointer to next segment
                                             */
};

struct start_list  {
                    int pointer;         /*  pointer to start seg  */
                    int column;          /*  column index of segment */
};
```

5.2 Structure for Frame-Dependent Information:

```
struct frame  {
          unsigned int ferrorstat; /*  system errors   */
          unsigned int ftod;       /*  time of day when picture
                                       was taken  */
```

```
        unsigned int fsequence;  /*  sequence number for
                                     picture  */
        unsigned int fpictype;   /*  the picture that was
                                     requested  */
        unsigned int fnumnodes;  /*  number of data entries
                                     found  */
        unsigned int ffirstnode; /*  pointer to first node  */
        unsigned int fnummatches;/*  number of matches whith
                                     expectations  */
                                 /*  used to be fnumfeats -
                                     still is for old system
                                     */
        unsigned int fnumedges;  /*  number of edges  */
        unsigned int ftabentries;/*  number of table entries */
};
```

6. PICTURE COMMAND STRUCTURE

6.1 Structure for Commands Passed Between MLDI and SSVA:

```
struct pictype  {
        unsigned int time;          /*  marker for current request
                                        */
        int camheight;              /*  height of camera or line
                                        flash above object */
        unsigned int bwindow;       /*  bottom position of window
                                        for camera  */
        unsigned int lwindow;       /*  left window  */
        unsigned int rwindow;       /*  right window  */
        unsigned int twindow;       /*  top window  */

        unsigned int flasht;        /*  type of flash (line of
                                        flood)  */
        unsigned int flashval;      /*  flash intensity  */
        unsigned int threshold;     /*  threshold  */
        unsigned int refadd;        /*  additive factor for
                                        reference frame (0-255)
                                        Mode  0  */
        unsigned int dadmode;       /*  mode of using the dad
                                        box:  0 = pixel-by-pixel
                                        threshold  */
                                    /*  (two pictures required),
                                        1 = "normal" single
                                        threshold mode */
```

```
        unsigned int reflshval;      /*  duration of flash for
                                         reference frame  */
                                     /*  dadmode 0 only, 0-15  */
        unsigned int reflasht;       /*  flash type of reference
                                         frame inage  */
        unsigned int opmode;         /*  mode of operation:  1 =
                                         operate default if no
                                         commands from above  */
                                     /*  0 = wait for new commands
                                         before proceeding  */
        unsigned int flashtest;      /*  flag for testing that
                                         flash fired.  0 = ignore
                                         flash test  */
                                     /*  1= perform flash test  */
        long int campos[6];              /* 6 words for the
                                         position of the camera
                                         */
        unsigned char picstatus;     /*  whether a normal request
                                         (0) or a number of frames
                                         */
                                     /*  to tell fsv to process
                                         for range in the given
                                         window  */
        unsigned char numwind;       /*  number of windows  */
        unsigned char startwind[10]; /*  up to 10 windows;
                                         defined by start and end
                                         */
        unsinged char endwind[10];
        unsigned int data_mode;      /*  0=read in 16 transitions;
                                         1=read in run centers  */
        unsigned int d_type;         /*  used only when data mode
                                         is 1. 0 = integer +
                                         fraction returned  */
                                     /*  1 = 16 center points
                                         returned  */
      unsigned int b_w_filter;       /*  noise filter for black to
                                         white transitions (0, 1,
                                         2, 3, pixels)  */
      unsigned int w_b_filter;       /*  noise filter for white to
                                         black transitions (0, 4,
                                         8, 12)  */
};
```

APPENDIX B. DEBUGGING OPTIONS

Interactive debugging options are provided for viewing intermediate results and for running the system in a simulated mode. To enable this mode, press any character on the supervisor keyboard while the system is running. A menu will appear on the screen with nine choices. They are explained in further detail below:

1. Vision system status - displays current vision system status with respect to presence of external systems.

2. Command stack dump - shows the contents of the vision command queue.

3. Command/answer history stack dump - shows approximately the 10 most recent question/answer pairs.

4. Change conditional printout variables - permits the user to change the level of detail in the printouts.

5. Reduce command stack and history stack - flushes the queue so that the control system does not have to wait for all of the queued commands to be executed.

6. Modify default thresholds - interactively changes the threshold for any object for either flood flash or line flash images.

7. Modify default picture - allows the user to interactively create a question and send it to either RCS or MHWS.

8. Change DADBOX mode - determines the mode in which the DAD framebuffer will be used.

9. Set up automatic sequence of questions - allows the user to specify a sequence of simulated commands to be posed to the vision system.

For commands 2 and 3 above, the format of the data is in unsigned decimal bytes. The bytes are in the protocol of input/output between the vision and control systems. Refer to Appendix A for the data structure format.

For command 4, the SUP printouts can be interactively changed to suit the user's needs. This provides a tradeoff between processing speed and debugging power. When command 4 is invoked,

each debug variable is printed out along with its current value. There are print statements in the program which are conditional on one of these debug variables; a "0", ignores the printout while a "1" permits it. The following debug variables are currently available:

1. ptrace - prints out a trace showing the order in which functions are called in the program.

2. debug - prints out almost everything.

3. debugm - prints out the returns in the main loop of the program.

4. debugvc - prints out the returns in the vision command section, a major part of the supervisor program.

5. debug589 - prints out useful information about the interaction between the control system and the vision system.

6. debugve1 - prints out lots of information concerning the acquisition of errors from the vision levels during picture processing.

7. debugve2 - prints out the errors from the vision levels during picture processing.

8. debugh - enables the history stack so that the most recent 10 commands and their answers can be stored for future use.

9. debug1 - enables the system to terminate when a drastic error occurs.

Another interactive feature, command 5, allows the command and history queues to be reduced so that they contain only the current command. This option is useful in cases where SUP has forced many simulated commands onto the queue before the control system initialized the 589 board. This option allows the control system's command to be processed immediately. Also, when the vision system is being operated in debug mode, this option allows different vision commands to be executed quickly.

Command 6 permits the user to change default thresholds for each part. When this command is activated, the first question chooses the type of flash, line or flood. The second question chooses

the part type. The user must enter the number of the part as it appears in the data structure, model (see Appendix A). The third question is the request for a new threshold value.

With command 7, the user can interactively send a question to the vision system. There are 5 choices of questions. Note that all flash values range between 1-10, and all range values and sector sizes are in millimeters. Question 1 calculates the centroid of one object in the field of view. Question 2 verifies that the part in the image is the expected part and also calculates its position and orientation. Question 3 calculates the range to the part in the image. Question 4 computes the centroid of four L.E.D.s in the field of view. This question is available but is not used in the AMRF. Question 5 verifies the occurrence of a part in a specified sector of a tray and also returns that part's position and orientation.

The next interactive feature, command 8, involves setting the thresholding used by the DAD hardware.

The last command, 9, allows the user to specify which sequence of questions will be posed to the vision system in simulated command mode.

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBSIR 87-3684	2. Performing Organ. Report No.	3. Publication Date December 1987

4. TITLE AND SUBTITLE

The NBS Vision System in the AMRF.

5. AUTHOR(S)

Marilyn Nashman and Karen Chaconas

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
U.S. DEPARTMENT OF COMMERCE
GAITHERSBURG, MD 20899

7. Contract/Grant No.

8. Type of Report & Period Covered

Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

National Bureau of Standards
Route #270 and Quince Orchard Road
Gaithersburg, MD 20899

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

This document describes the NBS Vision System in the AMRF. It discusses the objectives of the vision system and its applications in the factory environment. Since the vision system is a multi-processor system, each process is described according to its position in the vision hierarchy as well as to its particular logical and computational functions. The unique hardware used is discussed and its capabilities described. In addition, a guide to operations is included: This contains step-by-step directions for "bringing up" the system in either stand-alone mode or integrated mode. The interfaces between the individual processes of the vision system, as well as the interfaces between the vision system and other AMRF systems, are described. Finally, appendices are included which describe data structures, and debugging features.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

Automated Manufacturing Research Facility; hierarchical design; Material Handling Workstation; multi-processor system; Realtime Control System; realtime vision system.

13. AVAILABILITY

☒ Unlimited
☐ For Official Distribution. Do Not Release to NTIS
☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

48

15. Price

$11.95

USCOMM-DC 6043-P80