NBSIR 86-3393

A Low Level Robot Interface: The High Speed Host Interface

Marilyn Nashman

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
National Engineering Laboratory
Center for Manufacturing Engineering
Gaithersburg, MD 20899

June 1986

U.S. DEPARTMENT OF COMMERCE
NATIONAL BUREAU OF STANDARDS

NBSIR 86-3393

A LOW LEVEL ROBOT INTERFACE: THE HIGH SPEED HOST INTERFACE

Marilyn Nashman

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
National Engineering Laboratory
Center for Manufacturing Engineering
Gaithersburg, MD 20899

June 1986

U.S. DEPARTMENT OF COMMERCE, Malcolm Baldrige, *Secretary*
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Director*

Table of Contents

Figures

Tables

A LOW LEVEL ROBOT INTERFACE: THE HIGH SPEED HOST INTERFACE

Marilyn Nashman

Sensory-Interactive Robotics Group
National Bureau of Standards
Gaithersburg, MD 20899

ABSTRACT

This paper describes the High Speed Host Interface (HSHI) developed jointly by the Robot Systems Division at the National Bureau of Standards and the American Robot Corporation (AR). The High Speed Host Interface provides an interface between American Robot's software and hardware and the user's software at the lowest level of control--joint position, joint velocity, motor position, or motor velocity. It can operate at an update rate of 256 times per second or can be used at a cartesian level of control at an update rate of 7.5 times per second. The paper discusses the design of HSHI and its capabilities as well as some possible applications for such a system.

June 3, 1986

A LOW LEVEL ROBOT INTERFACE: THE HIGH SPEED HOST INTERFACE

Marilyn Nashman

Sensory-Interactive Robotics Group

National Bureau of Standards

Gaithersburg, MD 20899

1. Introduction

The HSHI is a low level interface designed to run on American Robot Corporations's Merlin Robot System, a six degree of freedom robot arm used in research as well as in industrial applications. Although this commercial equipment was purchased by the National Bureau of Standards and subsequently modified with the cooperation of the manufacturer, it is not the only equipment which might have been adapted to obtain equivalent performance. The six stepper motor driven axes are mounted on a base and consist of a waist joint, a shoulder joint, an elbow, a wrist rotate axis, a wrist flex axis, and a hand roll axis. (Figure 1) The robot, as supplied by the American Robot Corporation, includes not only the arm but a control system which can be programmed by using a teach pendant or ARSMART, a language developed by AR [3]. ARSMART permits the user to teach the robot arm points and to move from point to point in a defined order: it is used primarily for "pick and place" operations. While this capability is useful in many industrial applications, it does not lend itself to the area of intelligent machines. At the National Bureau of Standards' Robot Systems Division, we are interested in developing real-time sensory interactive control of robots[1,4]. A real-time sensory interactive control system must decide its output actions based on both the command goal and the sensory data that measure the state of the environment. (Figure 2) In addition these results must be output to the control system at fast

update rates to ensure an effective and stable response. It was with that concept in mind that the High Speed Host Interface was designed and developed.

The basic concept of the HSHI is to provide the user access to the lowest level of the control system through his own program control. Thus the user or host can control motor velocity and motor position at a rate of 256 updates per second, as well as being able to control robot position and orientation relative to a cartesian coordinate frame located at the center of the robot (Figure 3) at a rate of 7.5 times per second. The command update rate is defined to be the delay between the controller's receiving a command from the host and its execution of that command-- including the appropriate return of feedback status information. The types of feedback provided by the HSHI will be discussed further when the interface commands are described.

2. System Design and Protocol

The physical structure of the American Robot controller consists of a Motorola 68000 microprocessor and seven independent 6809 microprocessors [3]. These processors are housed in an electronic enclosure which includes a disk drive system that boots a Régulus Unix system and runs the HSHI software. Each of the 6809 processors is dedicated to a single task such as controlling a single motor. The software used with each of the 6809s is stored in PROM (Programmable Read Only Memory) chips located on the same circuit board as the 6809 which uses it. The HSHI software runs on the 68000 system and is stored on a floppy diskette. The 68000 system communicates with each of the 6809 microprocessors through shared memory over a Versabus. Using HSHI, these interprocessor communications and protocol are invisible to the host programs.

User programs are written in a combination of C language and 8086/8087 assembly language on a Multibus based CPM-86 operating system. The executable program is downloaded to an Intel 8086 microprocessor which is resident in AR's electronic enclosure. Data is passed between the Multibus and the Versabus systems via a Halversa Synergist board. (Figure 4)

The communication protocol between the host and the HSHI is implemented using the concept of a common memory area: a specific section of memory is designated as the common

memory area and is available to both systems for reading or writing commands and response feedback [4,5]. The common memory area has been divided into command and response buffers defined at specific addresses in the memory open to both processors. Table 1 lists the buffers used and their common memory definitions. The command buffers associated with each request will contain all pertinent data related to that request, e.g. desired position and orientation, motor velocities, joint positions, etc. The response buffers are written into by the HSHI and will contain current status information relevant to the type of information requested, e.g. cartesian position, motor position and velocity, cycle clock time, etc. The types of information that can be either read from or written to these addresses is predefined. The host program has write permission into all command buffers, while the HSHI program has read only access to those areas. Conversely, the HSHI program has write permission into the response buffers, while the host programs can only read those areas.

Unlike the control systems described by Barbera, Fitzgerald and Albus [2,4,5] in which reading and writing into the common memory area are permitted only at specific time intervals, the HSHI allows communication asynchronously. The HSHI is prepared to accept and act upon a new command as soon as it has completed processing its previous command. Depending on the type of command issued, communication can take place as often as every 4 milliseconds, or as infrequently as every 128 milliseconds. The host or the HSHI determine whether it can read or write from (into) the common memory area by examining a semaphore in the common memory handshake buffer. This flag can be set to either of two values: setting it to "HOST" implies that only the user program may read or write into common memory, and all status information is guaranteed to be updated as of the time tag associated with that data. When the flag is set to "HSHI", only the American Robot program has permission to read or write data. When either side has completed accessing the memory, the semaphore is set for the other user. Since all communication is based on a "command-response-command-response..." scenario, data integrity is insured. Neither command requests nor responses may be queued since the step-lock mode of operation is enforced by the protocol.

The remainder of the 16 byte handshake buffer is divided between the host and the HSHI. The host is responsible for setting a coded command byte which not only instructs the HSHI about the type of action it is to perform, but also provides a pointer to the appropriate data buffer in common memory from which to extract (or insert) the information it will require. In addition, the host sets the current cycle time (time of command) and updates the "total-commands-sent" value. Only after the relevant command data has been copied into the common memory buffer area, the command value set, and the remaining bookkeeping functions performed, does the user turn control to the HSHI by setting the "HSHI" flag in the handshake buffer. Table 1 lists the buffer areas associated with each of the commands. Table 2 describes the contents of the handshake buffer area, and Table 3 lists the defined commands and their coded command values.

The HSHI processor uses the handshake buffer to extract the command value and the relevant common memory data in order to perform the required task. Upon completion, it sets a "SUCCESS" or "FAIL" flag in the buffer's response word, as well as echoing the command value, starting cycle time and cycle time of command completion. Feedback information is written into the appropriate buffers, and the transaction is completed by setting the "HOST" flag.

3. Commands

There are thirteen codes which define requests to the HSHI processor; these can be divided into two categories: commands for action and requests for status. This section briefly describes *each of the command functions.* Appendix A describes the structure and format of the input or output associated with each command value.

Command 1 (Set Servo Parameters) is a request for resetting the robot arm's servo parameters. By issuing this instruction, the maximum acceleration, the maximum velocity, and the gain on each of the six motor axes can be individually adjusted in real-time. In this manner, the robot's performance can be observed and optimized for a particular application.

Command 2 (Command Cartesian Position) is used for positioning the robot arm at a point in cartesian space relative to the robot's base frame. The goal position and orientation refer to the position and orientation of the tool tip which is defined to be three and one half inches forward of the wrist axis. The X, Y, Z coordinates are expressed in inches, and roll, pitch and yaw are expressed in radians. If the goal point cannot be reached in a single execution cycle, the arm will continue to move at its maximum speed until that goal position is achieved. HSHI requires 128 milliseconds to perform the necessary computations required in decomposing a cartesian request to servo level commands. Note: Command 4 (Read Cartesian Position) must be issued before any cartesian level command if that command has been preceeded by a joint or motor level command.

Command 3 (Read Motor Position and Velocity) is a status request on the state of each of the six motors which can be executed at the rate of once every 4 milliseconds. The values returned in the pre-assigned memory buffers include the position of each of the motors expressed in motor encoder ticks, the velocity of each motor expressed in ticks per second, and the 6809 cycle time when each motor was read. A motor encoder tick is defined to be a unique incremental motor position equal to 1/2000 of a stepper motor shaft revolution. This command is implicitly performed whenever a motor position or motor velocity command is issued but is also provided as a stand-alone feature.

Command 4 (Read Cartesian Position) is a status request which returns the X,Y,Z coordinates of the tool tip in inches and its roll, pitch and yaw in radians. In addition, the current joint angles and motor position values are updated.

Command 5 (Command Joint Position) is used to position each joint in space; its inputs consist of six joint angles in radians. If the goal point commanded cannot be reached in a four millisecond

time interval, the arm will continue to move towards its goal at maximum velocity and acceleration.

Command 6 (Command Joint Velocity) requests specific joint velocities for each of the six robot joints. The velocities are expressed in radians per second. After a joint velocity command has been issued, the joints will continue to move at the commanded velocity until either another joint velocity command is received, a new motor velocity command is received (command 9), or a halt command (command 16) is issued. Because of the floating point arithmetic involved in this command, it is not considered to be high speed but can be executed once every 20 milliseconds. Motor position, motor velocity, joint position, joint velocity and the 6809 cycle count buffers are all updated as a result of the execution of this request.

Command 7 (Read Joint Position and Velocity) is a request for reading position and velocity status on each of the robot joints. The position of each joint is expressed in radians, and the velocity is expressed in radians per second. Because of the floating point computations involved, this command can only be executed every 20 milliseconds. In addition to returning the requested joint values, motor position, motor velocity, and 6809 cycle times are also updated. This command is built into commands 5 and 6, but is also provided as a stand-alone feature.

Command 8 (Command Motor Position) is the recommended fast mode of commanding motor position. The input for this command can either be expressed in radians or in motor encoder ticks. Command 8 differs from command 5 in that the HSHI can accept, act upon, and return status to the host within a four millisecond time interval. This "fast" mode of operation is made possible because the host program provides a utility program written for the 8087 Numeric Data Processor which converts joint angles expressed in radians (floating point variables) into motor encoder ticks (integers), or conversely, converts motor ticks into radians for more meaningful

status information. Appendix B describes the conversion formulas used for this purpose. At the completion of this command, motor position, motor velocity and 6809 cycle time are updated.

Command 9 (Command Motor Velocity) is the fast version of command 6, i.e. commanding motor or joint velocity. When the input to this command is expressed as joint velocities in radians, the host program converts these values into encoder ticks per second. The status information returned, motor position and motor velocity, can similarly be converted from encoder ticks to radians.

Command 10: Undefined

Command 11: Undefined

Command 12 (Set Cartesian Speed) is a request for setting the arm speed. It is issued prior to performing interpolation commands (commands 13 or 14). The value sent to HSHI represents the speed in inches per second that the arm is to travel.

Command 13 (Joint Interpolator) is a cartesian position request. When the host defines a goal point relative to the tool tip in cartesian space, a trajectory consisting of n subintervals, where n is a function of the requested arm speed, is planned by the HSHI. The path computed is based on the joint that has the farthest distance to travel: that joint is driven at top speed, and the remaining five joints are set so that all six joints will arrive at the goal point simultaneously. The processing time for computing each intermediate goal point in the interpolated path is 128 milliseconds. The advantage of this trajectory is that the arm will consistently be able to achieve its maximum velocity. If the host requests a position that the arm cannot reach, the "FAIL" flag

will be set in the common memory handshake buffer, and the trajectory motion will not be initiated. Note: Command 4 (Read Cartesian Position) must be issued before any cartesian level command if that command has been preceeded by a joint or motor level command.

Command 14 (Straight Line Interpolator) is a cartesian position request. The host program again supplies a goal point for the tool tip in cartesian coordinates relative to the robot's base frame. In this mode however, the HSHI plans a trajectory such that the tool tip will always travel in a straight line between its start position and its goal position. As with command 13, the processing time for each intermediate position is 128 milliseconds, and an error message will be returned if the goal point cannot be reached by the robot arm. Note: Command 4 (Read Cartesian Position) must be issued before any cartesian level command if that command has been preceeded by a joint or motor level command.

Command 15: Undefined

Command 16 (Halt Servoing) is a command to halt arm motion after either joint or motor velocities have been commanded. It effectively commands each motor to a velocity of zero motor encoder ticks per second without the host having to supply input parameters.

4. Uses for the High Speed Host Interface

The unique capability of the HSHI to receive and process servo level commands in a four millisecond time interval makes it an especially useful tool in implementing certain low level control system tasks. In [6], Featherstone presents algorithms for calculating robot joint variables from the position and velocity of the end effector and vice versa. His algorithms translate the path of the end effector into joint or motor values which are the values under control. The success of this method is very dependent on the ability to perform the required calculations in real-

time and to rapidly update feedback information in the form of joint velocities to the robot. This paper does not address the problems associated with real-time computation, but by taking advantage of hardware math accelerators, parallel processing, optimized code, and look up tables for trigonometric functions, updated commands can be computed and delivered to the HSHI , and motor position and velocity status returned to the host in the minimal four millisecond time interval. Figure 5 describes the recommended optimal use of the HSHI. Portions of Featherstone's algorithms have been implemented on the AR Merlin robot in the Robot Systems Division's robot laboratory using the HSHI.

A hierarchical control system based upon the HSHI is a logical extension of this project [7]. The lowest level of the hierarchy, the HSHI, would accept position or velocity commands in motor coordinates, and return motor position and velocity status. The next level could be a low level control and would provide real time position or velocity commands in motor encoder ticks to the HSHI. Algorithms such as those developed by Featherstone [6] would be implemented at this level. The commands sent to the arm could be developed by interpreting trajectories provided by a still higher control level or by servoing on sensory feedback.

5. Conclusion

The High Speed Host Interface developed jointly by the Robot Systems Division at the National Bureau of Standards and the American Robot Corporation provides access to the servo level of control by accepting position or velocity commands in motor or joint coordinates and returning relevant status information 256 times per second. At a slightly higher level, commands expressed in cartesian coordinates can be processed 7.5 times per second. A handshake protocol and common memory area have been designed and implemented which insure proper communication between the host and HSHI programs. A library of C language subroutines and 8086/8087 assembly language routines (Appendix C) has been developed to test and implement the features of the HSHI.

References

1. J. Albus, A.Barbera, M.L.Fitzgerald, M.Nashman, "Sensory Interactive Robots", Presented at the 31st General Assembly of the International Institution for Production Engineering Research, September,1981.
2. J. Albus, C. McLean, A. Barbera, M.L. Fitzgerald,"An Architecture For Real-Time Sensory Interactive Control of Robots in a Manufacturing Facility", IFAC Information Control Problems in Manufacturing Technology, 1982.
3. American Robot Corp., "Merlin Robot System, Operator and User Guide", Manual #SMT-2.0-0184, Revision 2.1,February,1984.
4. A.Barbera, M. Fitzgerald, J. Albus,"Concepts for a Real-Time Sensory Interactive System Architecture", Proceedings of the Fourteenth Southeastern Symposium on System Theory, April,1982.
5. A. Barbera, M. Fitzgerald, J. Albus, L. Haynes, " A Language Independent Superstructure for Implementing Real-Time Control Systems",1984.
6. R. Featherstone,"Position and Velocity Transformations Between Robot End-Effector Coordinates and Joint Angles", International Journal of Robotics Research, Vol 2, No. 2, Summer,1983.
7. E. Kent, "Space Domain Control with Fitts'-Law Functions and Separation of Translation and Orientation Trajectory Spaces", 1984.
8. J. Toth, "Merlin Universal Controller High Speed Host Interface", American Robot Corp., March 1986.

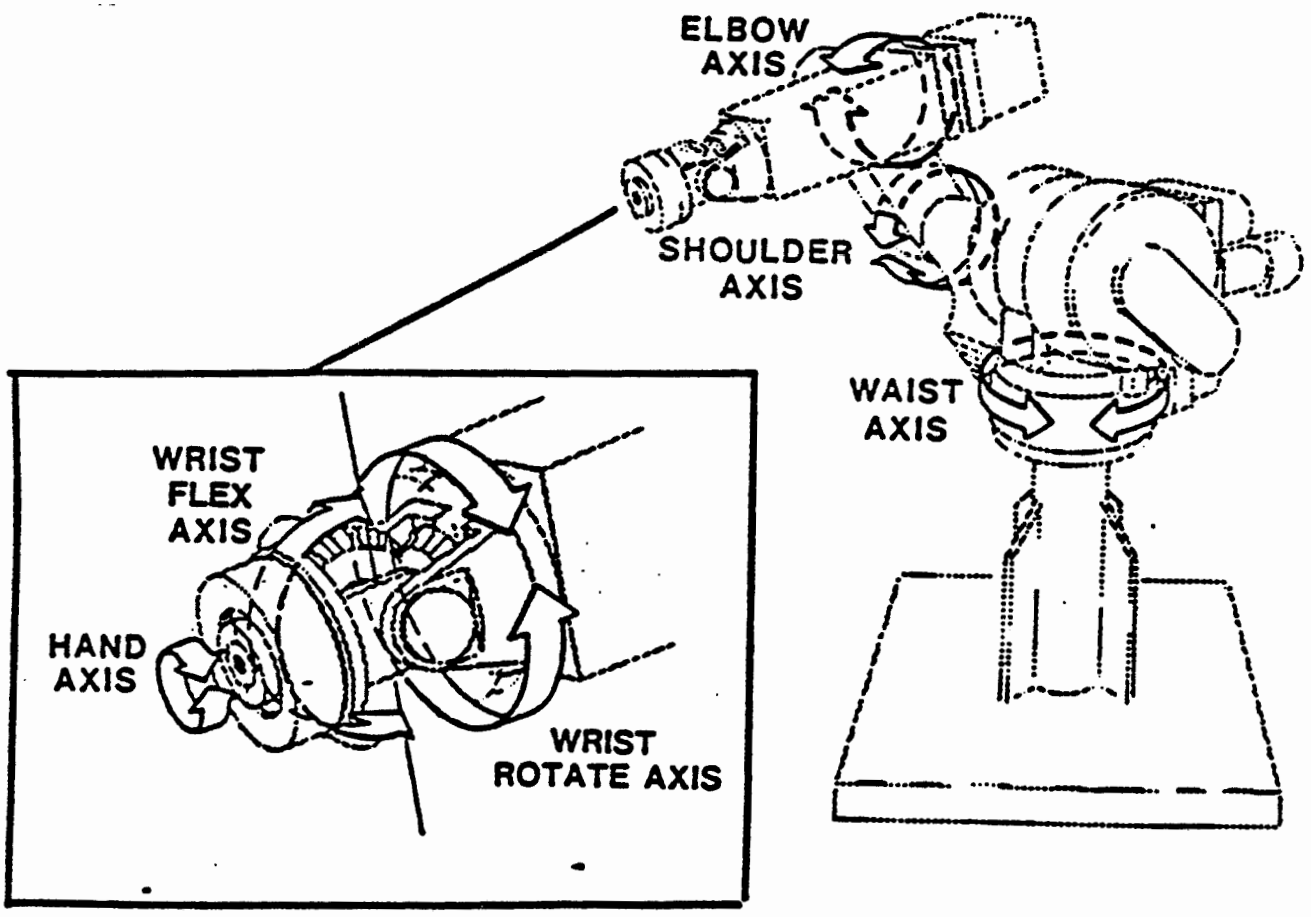

Figure 1: The American Robot Merlin Arm

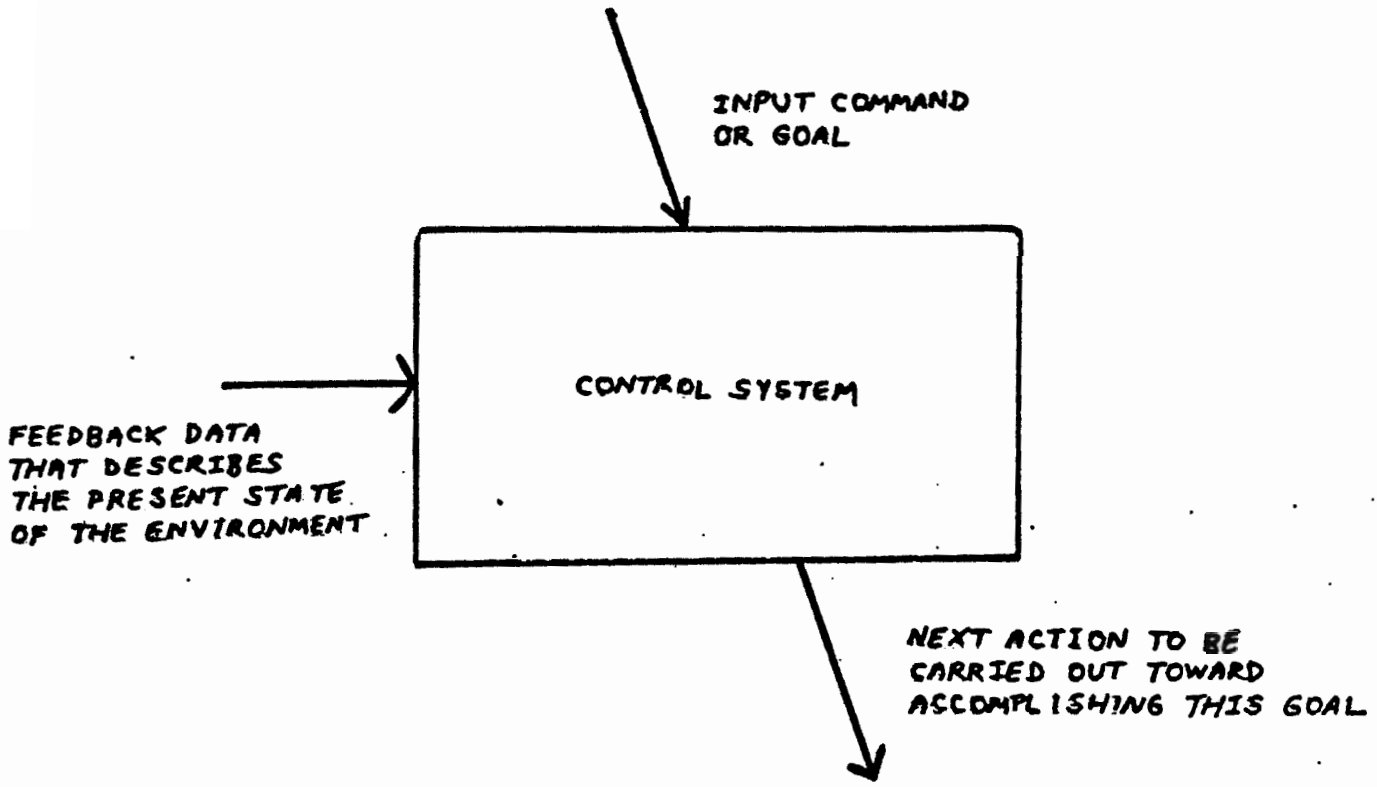

Figure 2: Control System Inputs and Outputs

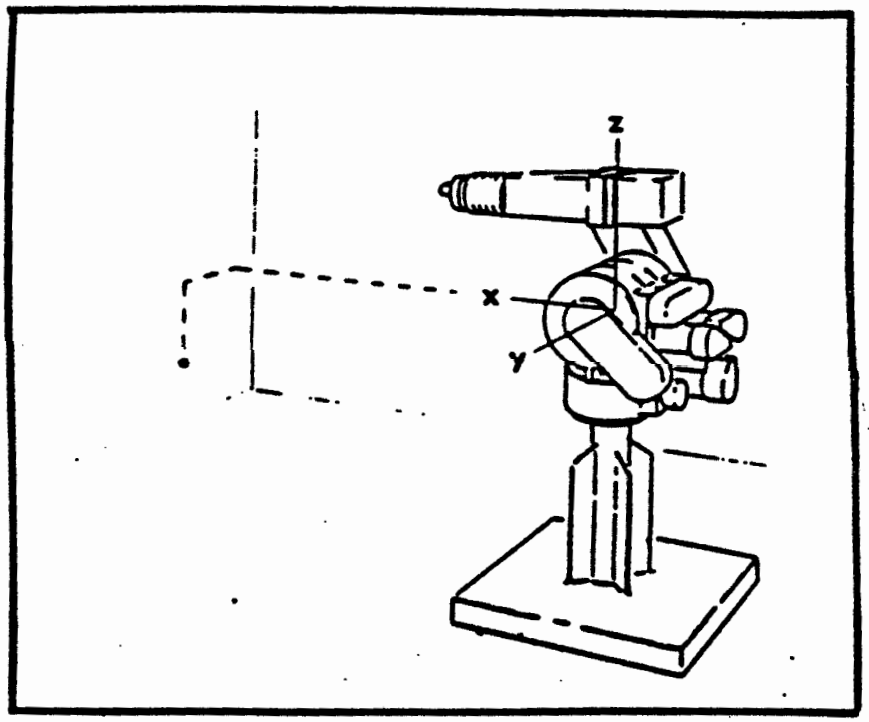

Figure 3: The Robot Origin and Coordinate Axes

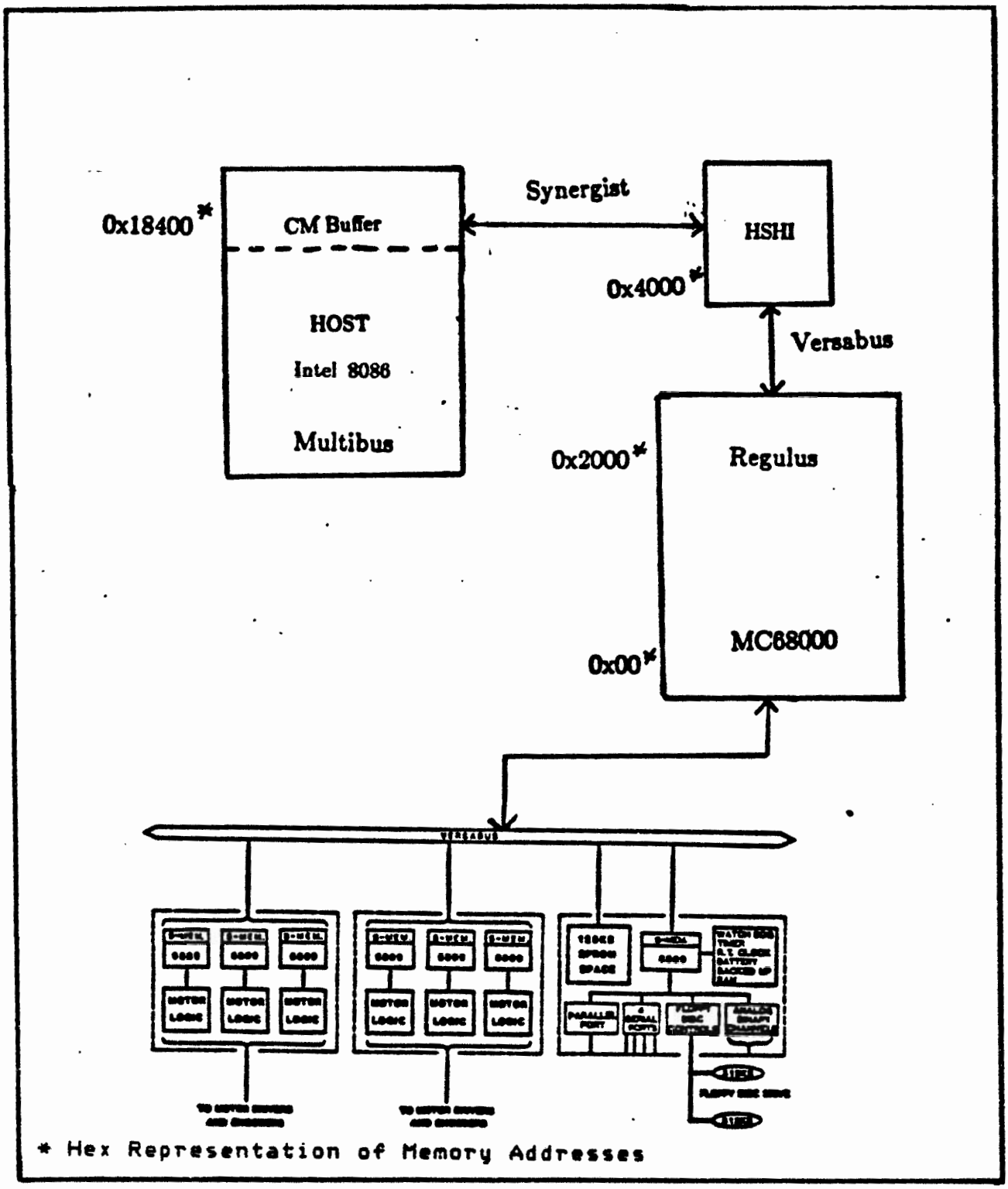

Figure 4: High Speed Host Interface Architecture

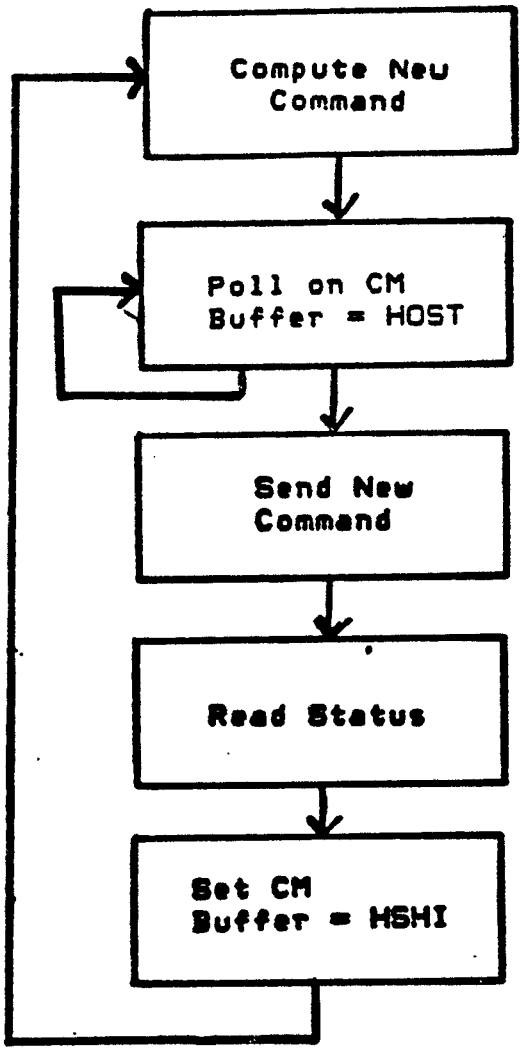

Figure 5: Optimal Host Program Design

PROTOCOL BUFFER	ADDRESS	NUMBER OF BYTES	
Handshake	18400 hex	16	
COMMAND BUFFER NAME	ADDRESS	NUMBER OF BYTES	COMMAND #
Servo Parameters	18410 hex	60	1
Cartesian Position	18460 hex	24	2, 13, 14
Interpolator Speed	18480 hex	4	12
Joint Position	184A0 hex	24	5
Joint Velocity	184C0 hex	24	6
Motor Position	184E0 hex	24	8
Motor Velocity	500 hex	24	9
RESPONSE BUFFER NAME	ADDRESS	NUMBER OF BYTES	COMMAND #
Cartesian Position	18600 hex	24	4
Joint Position	18620 hex	24	7
Joint Velocity	18640 hex	24	7
Motor Position	18660 hex	24	3, 8, 9
Motor Velocity	18680 hex	24	3, 8, 9
Cycle Time	186A0 hex	24	3, 8, 9

Table 1: Common Memory Map

Bytes 1 and 2 : System Cycle Clock
Bytes 3 and 4 : Total Number of Commands Sent
Bytes 5 and 6 : Current Cycle Number
Byte 7 : Buffer Ready Flag
Byte 8 : Command
Bytes 9 and 10 : Echoed Command
Bytes 11 and 12: Echoed Cycle Number
Bytes 13 and 14: End Cycle Number
Bytes 15 and 16: Response Code

Table 2: Common Memory Handshake Buffer

#	Action
1	: Set Servo Parameters
2	: Command Cartesian Position
3	: Read Motor Position and Velocity
4	: Read Cartesian Position
5	: Command Joint Position
6	: Command Joint Velocity
7	: Read Joint Position and Velocity
8	: Command Motor Position With Status Returned
9	: Command Motor Velocity With Status Returned
10	: Undefined
11	: Undefined
12	: Set Speed For Cartesian Moves
13	: Joint Interpolator
14	: Straight Line Interpolator
15	: Undefined
16	: Halt Joint Angle Servoing

Table 3: HSHI Commands

Appendix A

The data structures and variable formats used in each of the HSHI commands are listed in this section. In accordance with 8086 definitions, a byte is defined to be 8 bits, a character to be 1 byte, an integer to be 2 bytes and a long integer to be 4 bytes. Floating point numbers are 4 bytes long and are represented in IEEE format.

Command Number: 1

Description : Set Servo Parameters -- maximum velocity, acceleration and gain for each of the six motors.

Data Structure:

```
struct servo_motor{
  long int maxacc;      /* in revolutions per second squared*/
  long int maxvel;      /* in revolutions per second */
  long int gain;      /* a servo-loop multiplier */
};

struct servo_param{
  struct servo_motor motor1;
  struct servo_motor motor2;
  struct servo_motor motor3;
  struct servo_motor motor4;
  struct servo_motor motor5;
  struct servo_motor motor6;
};
```

Range of values: 0 <= maxacc <= 32
0 <= maxvel <= 12
0 <= gain

Default values: maxacc = 8
maxvel = 12
gain = 4

Command Number: 2

Description : Command Cartesian Position

Data Structure:

```
struct cart_pos{
  float c_xpos;       /* x,y,z position in inches */
  float c_ypos;
  float c_zpos;
  float c_roll;      /* roll,pitch and yaw in radians*/
  float c_pitch;
  float c_yaw ;
} ;
```

Calibration position is defined at 38.124,-11.9,0,0,0,0

Command Number: 3

Description : Read motor status -- position, velocity and time cycle at which those readings were obtained.

Data Structure:

```
struct r_mpos{   /* Motor position is expressed in encoder ticks
  long int r_m1;      2000 ticks per motor revolution.*/
  long int r_m2;
  long int r_m3;
```

```
  long int r_m4;
  long int r_m5;
  long int r_m6;
} ; Read Only Buffer-- updated by AR after each motor
     velocity or position command is issued
```

Calibration position is defined at 0,0,0,0,0,0

```
struct r_mvel{    /* Motor velocity in encoder ticks */
  long int r_vm1;
  long int r_vm2;
  long int r_vm3;
  long int r_vm4;
  long int r_vm5;
  long int r_vm6;
} ;  Read Only Buffer--updated by AR after each motor
     velocity or motor position command is issued

struct r_mcyc{    /* Incremental counter set by the 6809 boards
  long int r_mcyc1;  every 4 milliseconds-- useful for checking
  long int r_mcyc2;  timing of commands */
  long int r_mcyc3;
  long int r_mcyc4;
  long int r_mcyc5;
  long int r_mcyc6;
} ;  .  .      (Read Only Buffer)
```

Command Number: 4

Description : Read Cartesian Position

Data Structure:

```
struct r_cart_pos{
  float r_x;      /* x,y,z in inches*/
  float r_y;
  float r_z;
  float r_roll;   /* roll, pitch and yaw in radians*/
  float r_pitch;
  float r_yaw;
  } ;     (Read Only Buffer-- updated only after command 4
   is issued)
```

Command Number: 5

Description : Command Joint Position

Data Structure:

```
struct joint_pos{
  float c_j1;     /* Joints 1 -6 expressed in radians */
  float c_j2;
  float c_j3;
  float c_j4;
  float c_j5;
  float c_j6;
  } ;
```

Calibration Position: 0,0,0,0,0,0

Command Number: 6

Description : Command Joint Velocity

Data Structure:

```
struct j_vel{     /*Joints 1 - 6 expressed in radians/second */
  float c_vj1;     Note: Velocity commanded will continuously
  float c_vj2;     be sent to the arm until command 16
  float c_vj3;     is issued
  float c_vj4;
  float c_vj5;
  float c_vj6;
  } ;
```

Command Number: 7

Description : Read Joint Status

Data Structure:

```
struct r_jpos{    /* Joint position returned in radians*/
  float r_j1;
  float r_j2;
  float r_j3;
  float r_j4;
  float r_j5;
  float r_j6;
  } ;      Read Only Buffer updated when joint position
   or joint velocity are commanded.

struct r_jvel{    /*Joint velocity returned in radians/sec*/
  float r_jv1;
  float r_jv2;
  float r_jv3;
  float r_jv4;
  float r_jv5;
  float r_jv6;
  } ;        Read Only Buffer- updated when joint position
     or joint velocity are commanded)
```

Command Number: 8

Description : Command Motor Position

Data Structure:

```
struct m_pos{     /* Motor positions expressed in encoder ticks */
  long int c_m1;
  long int c_m2;
  long int c_m3;
  long int c_m4;
  long int c_m5;
  long int c_m6;
} ;
```

Command Number: 9

Description : Command Motor Velocity

Data Structure:

```
struct m_velocity{  /* Motor velocity expressed in ticks/sec */
  long int c_vm1;     Note: Velocity commanded will continuously
  long int c_vm2;     be sent to the arm until command 16
  long int c_vm3;     is issued
```

```
    long int c_vm4;
    long int c_vm5;
    long int c_vm6;
  } ;
```

Command Number: 12

Description : Set Speed for commands 13 and 14 (interpolation commands)

Data Structure:

```
  struct cart_vel{
    float rate;     /* speed in inches per second. */
    } ;
```

Range of input: .009 <= rate <= 30.0

Default value: 5 inches per second.

Command Number: 13

Description : Joint interpolator.

Data Structure:

```
  struct cart_pos{
    float c_xpos;   /* x,y,z position in inches */
    float c_ypos;
    float c_zpos;
    float c_roll;   /* roll,pitch and yaw in radians*/
    float c_pitch;
    float c_yaw ;
  } ;
```

Command Number: 14

Description : Straight line interpolator.

Data Structure:

```
  struct cart_pos{
    float c_xpos;   /* x,y,z position in inches */
    float c_ypos;
    float c_zpos;
    float c_roll;   /* roll,pitch and yaw in radians*/
    float c_pitch;
    float c_yaw ;
  } ;
```

Command Number: 16

Description : Halt servoing on joint or motor velocity commands.

Data Structure: None

Appendix B

The following C code defines the formulae used to convert joint angles, expressed in radians, into motor encoder ticks and vice-versa. The constants defined are particular to the American Robot Merlin Robot used in the Sensory-Interactive Robot Lab. The actual conversion code used is written in 8087 assembly language.

Radians to Motor Encoder Ticks

```
rad_to_ticks(mr)
float mr[]; {              /* Array  mr originally contains 6 joint angles
                             expressed in radians.  At the completion of the
                             routine, the array will contain the equivalent
                             motor encoder readings in motor tick units */

double temp;

temp = mr[4] * 1.2;

mr[5] = mr[3] + temp - mr[5];

mr[4] = mr[3] - temp;

mr[0] *= NEG_MAIN_RATIO ;

mr[1] *= MAIN_RATION ;

mr[2] *= NEG_MAIN_RATIO ;

mr[3] *= RATIOW ;

mr[4] *= RATIOW ;

mr[5] *= RATIOW ;

}
```

Motor Encoder Ticks to Radians

```
ticks_to_rad(mr)
float mr[]; {          /* Array mr contains 6 motor encoder readings.
                             The array will contain joint angles
                             in radians upon exit */

double ratio0;
double ratio2 ;
double ratio1 ;
```

```
ratio0 = ratio2 = -1.0* MAIN_RATIO;
ratio1 = MAIN_RATIO;

mr[0] = mr[0]/ratio0;

mr[1] = mr[1]/ratio1;

mr[2] = mr[2]/ratio2;

mr[3] = mr[3]/(K*WRGR);

mr[4] = mr[4]/(K*WRGR);

mr[5] = mr[5]/(K*WRGR);

mr[5] = 2.0 * mr[3] - mr[4] -mr[5];

mr[4] = (mr[3] - mr[4]) * RATIO4L;
}
```

The constants used in this conversion are:

```
#define MAIN_RATIO 15278.875
                                /* MAIN_RATIO = (ET*TGR)/2*PI  where :
                                   ET = 2000 encoder ticks per rev.
                                   TGR = transmission gear ratio = 48
                                   2 PI = 6.283285  */
#define NEG_MAIN_RATIO -15278.875
#define WRGR 24.0               /* wrist gear ratio */
#define K 318.310               /* number of motor ticks per radian */
#define RATIOW WRGR*K
#define RATIO4L 0.833333333     /* gear ratio constant used in
                                   conversion of joint 4 from encoder ticks
                                   to radians */
```

Appendix C

Following is a brief description of the major subroutines contained in the HSHI library written by the Sensory-Interactive Robotics Group at the National Bureau of Standards.

init_common_memory();
This routine initializes the common memory handshake buffer. It must be called before any commands are sent to the HSHI.

format_command(command number, buffer_address);
This subroutine fills the common-memory areas with the appropriate values as indicated by "command number". Data to be sent to the HSHI should be in "buffer_address". If a "read data" command is issued, (e.g. read motors), set buffer address to NULL (or any dummy value). Data values are converted to 68000 compatible format via a call to wordswap() which is described below.

EXAMPLE:
To command cartesian position 38.124,-11.9,0,0,0,0

```
float home[6];
home[0] = 38.124;
home[1]= -11.9;
home[2]=home[3]=home[4]=home[5]=0.0;
command = 2;
format_command(command,home);
```

dump_cm_buffer();
Prints the current contents of the common memory handshake buffer. Last command issued, number of commands issued, buffer_ready flag, and response bits can be read.

radtotic(floating pt array,long integer array);
8087 subroutine to convert six joint positions or velocities to motor ticks for faster processing. Input array (radians) in floating point format is converted to long integer array (also six values) in ticks.

tictorad(floating pt array,long integer array);
The converse of radtotic-- motor encoder ticks in long integer format are converted to joint angles in radians. Both input and output arrays must contain 6 elements.

waitbuf()
Assembly language routine to be called before reading in requested status feedback. Makes sure HSHI has completed transferring the data to common memory before the host attempts to read.

wordswap(buffer,bufsize);
Before and after each data transfer between the 68000 and the 8086 systems, this routine is called to swap high and low order 16 bit quantities in 32 bit floating point numbers or long integers. Note: this routine destroys the original contents of its input buffer. Wordswap is called automatically by library subroutines and need never be called by the user.

```
read_cp(cartpos)
   struct r_cart_pos *cartpos;
```

Command 4 (read cartesian position) is implicitly sent to the interface (using format_command). Cartesian position data is then transferred from the common-memory area into the local buffer cartpos. Routine wordswap() is used to format the data into 8086 form.

```
read_motors(motorpos,motorvel,motcyc)
    struct r_mpos *motorpos;
    struct r_mvel *motorvel;
    struct r_mcyc *motcyc;
```

After command 3 (read motor status) is sent to the interface, this routine transfers the appropriate common memory areas into the local buffers motorpos,motorvel, and motcyc. wordswap() is used to format the data into 8086 form.

```
getmotorpos(motorpos,radval)
   struct r_mpos *motorpos;
   float radval;
```

Transfer motor position from common memory buffers into array motorpos and convert it to radians. Result is stored in array "radval".

```
get_cyc_no(c_buf)
   struct r_mcyc *c_buf;
```

Transfer the 6809 motor cycle time to buffer c_buf. The 6809 cycle number is updated every 4 milliseconds whenever an action command is issued.

```
red_joints(jointpos,jointvel)
   struct r_jpos *jointpos;
   struct r_jvel *jointvel;
```

After command 7 (read joint status) is issued , this routine transfers the appropriate common memory areas into the local buffers jointpos and jointvel. wordswap() is called to format the data into 8086 form.

```
read_response();
```

This routine polls for the completion of the last issued command and returns the response word set by the 68000 program. A negative value returned indicates an error.

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

U.S. DEPT. OF COMM. **BIBLIOGRAPHIC DATA SHEET** *(See instructions)*	1. PUBLICATION OR REPORT NO. NBSIR 86-3393	2. Performing Organ. Report No.	3. Publication Date JUNE 1986

4. TITLE AND SUBTITLE

The High Speed Host Interface

5. AUTHOR(S)

Marilyn Nashman

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

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

Final

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

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 paper describes the High Speed Host Interface (HSHI) developed jointly by the Robot Systems Division at the National Bureau of Standards and the American Robot Corporation (AR). The High Speed Host Interface provides an interface between the manufacturer's software and hardware and the user's software at the lowest level of control--motor position and velocity. It can operate at an update rate of 256 times per second or can be used at a cartesian level of control at an update rate of 7.5 times per second. The paper discusses the design of HSHI and its capabilities as well as some possible applications for such a system.

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

common memory; interface; low level control; microprocessor; real time sensory interactive control; robot controller; servo level control

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

30

15. Price

$ 9.95