RCS: The NBS Real-Time Control System

Anthony J. Barbera
National Bureau of Standards
Washington, District of Columbia

M.L. Fitzgerald
National Bureau of Standards

James S. Albus
National Bureau of Standards

Leonard S. Haynes
National Bureau of Standards

Abstract

The National Bureau of Standards, Industrial Systems Division designed the Real-time Control System where high-level goals are decomposed through a succession of levels, each producing strings of simpler commands to the next lower level. The bottom level generates the drive signals to the robot, gripper, and other actuators. Each control level is a separate process with a limited scope of responsibility, independent of the details at other levels, thus providing a foundation for future modular, "plug compatible" hardware and software for robots and real-time sensory interactive control applications. To aid in specifying the required task decomposition and task processing, a programming language and program development environment were implemented. Programs at each control level are expressed as state tables, and the programming environment permits the generation, editing, emulation, and evaluation of these state tables. The control system is completely interactive, allowing the system to run freely, or be single-stepped to any level of detail.

1. INTRODUCTION

The theory behind the NBS Real-time Control System (RCS) has been under development for almost a decade. References 1 to 5 document that research. A paper describing the implementation details of RCS is in preparation. The following paper describes the users view of the control system, including an example of a current robot application. Figure 1 shows the major components of RCS. These components can be described in terms of a user entry mechanism, hierarchically executing control levels, common memory, communications, and a diagnostic module.

1.1 DATA, STATE TABLE, AND PROGRAM EDITORS

On the right side of figure 1 are the Data Form Editor, State Table Editor, and Program Editor. These editors represent three levels at which a user can interact with RCS. For entry of, or changes in data, a user uses the data forms system. Using a set of forms, part descriptions, names, and locations of machines, fixtures, etc., are entered. The data so entered is entirely distinct from the application algorithms. Separating the task description from the data specification helps make the control algorithms directly transferrable to many different workstations and applications by a change in the data base description of the parts and worksite. The data forms are detailed in section 3.2.

In RCS, application algorithms are specified via state tables, which primarily perform task decomposition, and procedures called by these state tables which primarily perform computation. For a given application, most programming will be done via the state tables alone. This will control the sequence of, and conditions under which various procedures will be called. The State Table Editor provides a convenient tool for editing state tables. State tables are described in detail in section 2.8.

For major changes to RCS application codes it may be necessary to add or modify procedures as well. A Program Editor is provided for this purpose. The specific language in which these procedures are written is called SMACRO, a general purpose, "full power," programming language. SMACRO is discussed in section 2.8.

The details of the editors are very conventional and will not be discussed further in this paper. An RCS users manual which will contain these details is in preparation.

1.2 CONTROL LEVELS

Complex control systems are built in RCS by implementing hierarchical control levels. Each control level represents a well-defined clearly-bounded control function with a small number of inputs and outputs. Commands input at the highest level are decomposed into sequences of subtasks which are passed as commands to the next lower level in the hierarchy. This same procedure is repeated at each level until, at the bottom of the hierarchy, a set of outputs is generated to actuators, interlocks, and signal lines to cause the necessary external response. The complexity at any level in the hierarchy is held within manageable limits, regardless of the complexity of the entire structure. Currently, for robot control, three robot

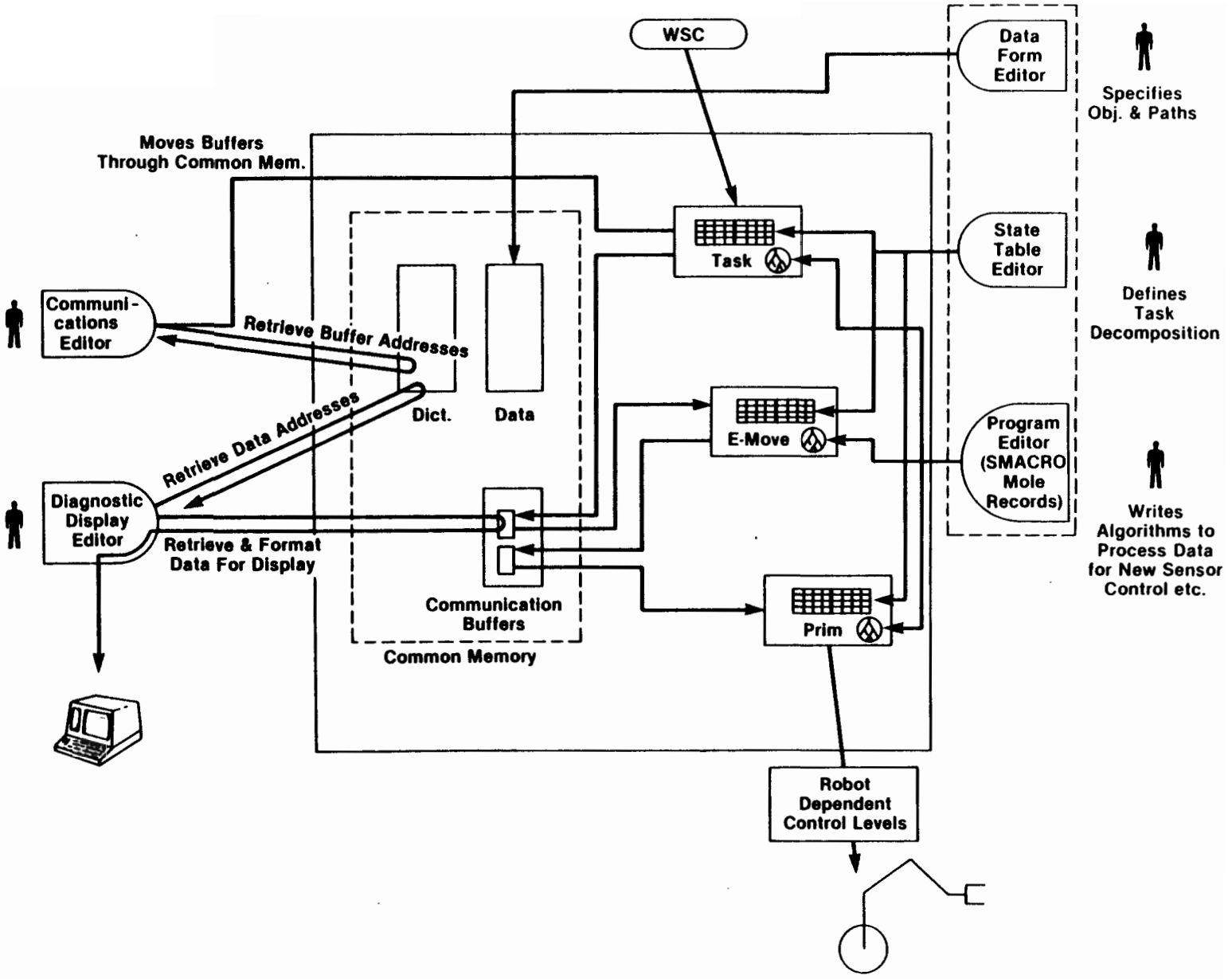


Figure 1. Real-Time Control System

independent control levels have been implemented and fully tested. These three levels, called the TASK, E-MOVE, and PRIMITIVE levels are discussed in section 2.2. Below the PRIMITIVE level there is a generic interface to any robot, and below that there are two robot dependent control levels. The robot dependent levels of the control system are beyond the scope of this paper.

1.3 COMMON MEMORY

Each of the functional elements of RCS can access a common memory. The system dictionary, forms systems data, and communication buffers reside in this common memory.

1.3.1 Data Dictionary

The system dictionary is by far the largest part of the common memory, storing all of the variables, procedure, and state tables within the system. Named variables, named procedures, named state tables, lists of variables, and lists of procedures, are all stored in the data dictionary. The user can view or change the value of any variable, or list of variables, can call for the execution of any procedure, a list of procedures, or execute one cycle of any state table, all by typing the relevant variable, procedure, list, or state table name. The data dictionary is strictly offline, and is not referenced during the actual execution of control levels. The data dictionary will be discussed further in section 3.3.

1.3.2 Data

The part and location data which the user enters into the forms is tabularized by the forms systems and stored in the data portion of common memory. None of the state tables or procedures contain data, all data is referenced from the common memory online just before the data is actually needed.

1.3.3 Communication Buffers

The communication buffers are the common memory buffers used for all transfer of data betweeen every control level, every sensory system, and the interfaces to the robot, gripper, and other external systems.

1.4 COMMUNICATIONS EDITOR

As described above, all intercommunication occurs via buffers stored in common memory. Data source and destinations are specified via the communications editor by giving the variable list name of the data source and its location, and the variable list name of the destination and its location. The communication system executes synchronously each and every cycle. The Communications Editor is further decribed in section 3.1.

1.5 DIAGNOSTIC SYSTEM

RCS provides a user friendly, powerful Diagnostic System. Because all data communication occurs through common memory every control cycle, the current state of the machine is available from the common memory at any time. By accessing this data, the Diagnostic System can provide real-time readout of values, transfer any values to external systems, etc. The Diagnostic System also makes a log of the status of the communications buffers and can actually step the system through these previous machine states to locate the source of a problem. Included in the Diagnostic System

is a very powerful Display Editor and Graphics Interface which supports real-time plotting of variables, vectors, etc. The Diagnostic System is discussed in section 4.

2. CONTROL LEVELS

At the highest level of RCS, input commands provide an overall goal for the system that is reached through a sequence of subgoals passed down to the next lower level, which in turn, further decomposes those subgoals to be executed by lower control levels. The output of each level depends on the command being processed, the present state of the environment as indicated by sensory feedback data, the internal state of the control level, and status feedback from the lower level.

2.1 STRUCTURE OF A CONTROL LEVEL

The atomic unit within a control level is a functionally bounded module, as shown in figure 2. This module consists of inputs, a process, and outputs. Each functionally bounded module is given a name, shown in quotes to indicate it is an ASCII string, and the module can be executed at any time by typing that name on the terminal. Input variables can be changed by typing <variable name=new value> and the value of any variable can be interrogated by typing <name ?>. Functionally bounded modules are executed in fully compiled machine code form; hence, the interactive capability does not severely impair efficiency.

Functionally bounded modules can be grouped within an executing owner with its own name, as shown in figure 3. Typing the name of the owner causes the functionally bounded modules to be executed in the order listed. The essential concept is to keep modules distinct even when they will be executed together.

Executing owners are grouped into control levels. The structure of a control level, shown in figure 4, is generic and every level has the identical structure. A control level consists of a preprocess function, a state table function, and a postprocess function. Every control level is executed every cycle. The preprocess functions are used to combine and/or convert incoming data into a format suitable for the rest of the level. The postprocessing functions convert output data into formats required for other systems. Decision processing in a control level is done by a state table (ref 2). As shown in figure 4, the left side of the table is a list of conditions which test the relevant state of the world as determined from the input variables and internal machine state. Exactly one line of this table must match with the relevant state of the world. In this case, the ouput procedures listed on the right side of that state table line are executed. State tables will be described in more detail later. For the present, we note several characteristics of state tables which are not present in procedural software programs.

a. The NBS Robot Control System does not require interrupts. The state table inputs are examined each cycle. If a particular input condition requires immediate attention, this can be reflected in the state table, and because the state table is examined every cycle it is guaranteed that whatever action is required will be initiated within a maximum of 1 cycle time. It would of course be possible for a user to write conventional software which produced the same result, also without interrupts, but the NBS Control System structures hides much of the necessary detail from the user, and ensures consistency in complex application code.

b. A large percentage of the code for robot applications will effect recovery from various undesirable events. Using state tables organized into control levels as described, additional situations can be handled by adding lines to the state table. With conventional software a user must first figure out where in the

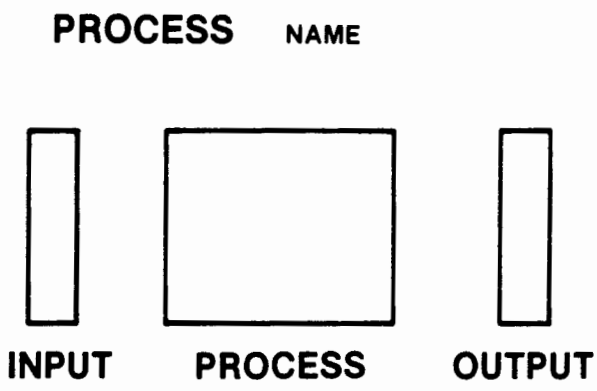


Figure 2. Functionally Bounded, Named Module
Functionally Bounded Modules are Independently Executable

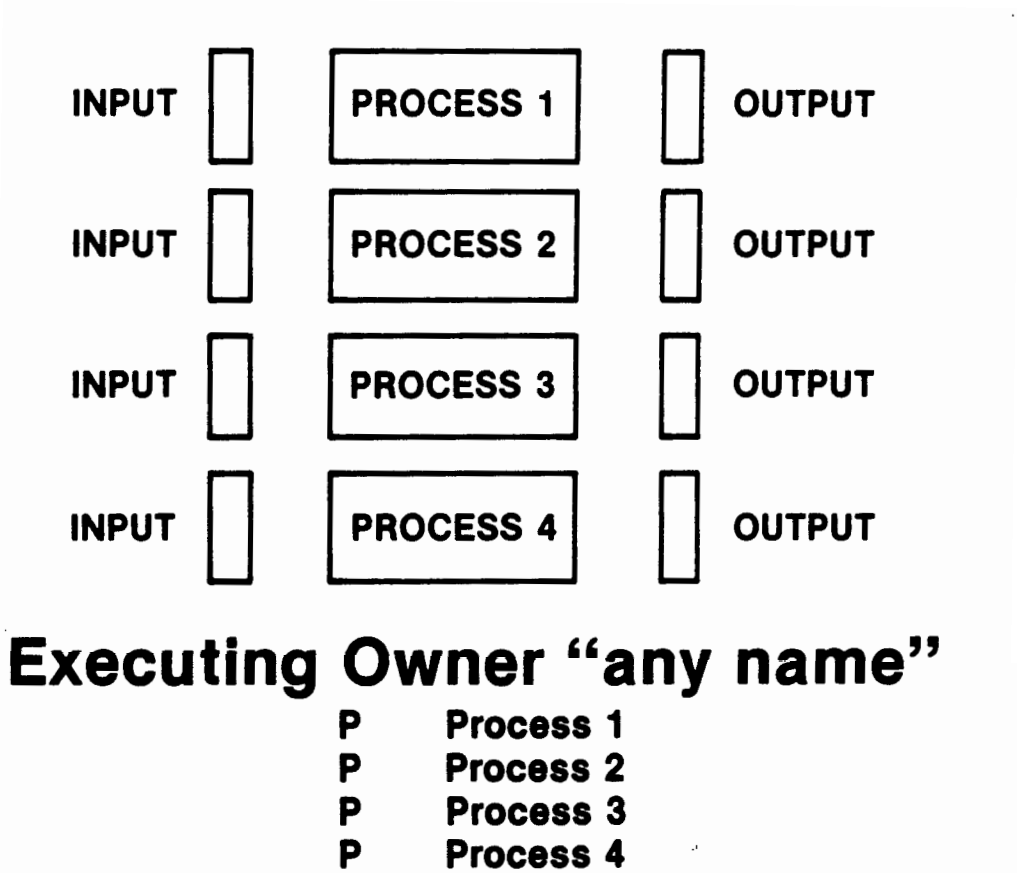
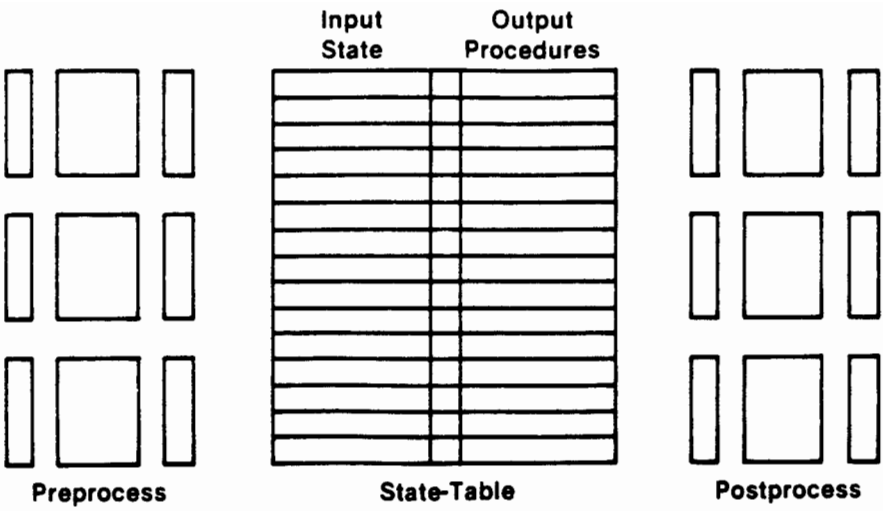


Figure 3. Executing Owner "any name" Procedures are Combined Together Under the Owner Name Yet Remain Distinct, Understandable, and Independently Executable



Executing Owner TASK LEVEL

- eo Preprocess
- eo State-Table
- eo Postprocess

THE ENTIRE CONTROL LEVEL IS EXECUTED ONCE EACH CYCLE.

Figure 4. Structure of a Control Level

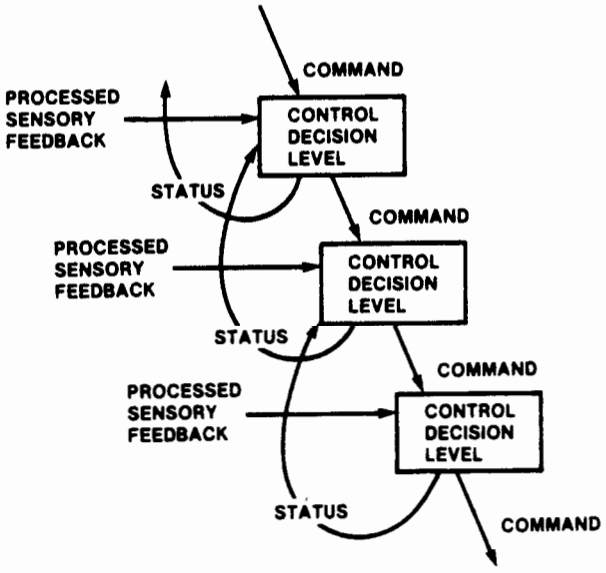


Figure 5. Hierarchical Control Levels Linked Together to Form a Complex Control System

code this particular situation may occur, and changes and/or patches may have to be made in many places in the code. Stated differently, state tables isolate those operations that are required for each specific situation, and let the application designer think in this high level manner - what to do in each case, independent of if then, loop, or other software control structures.

c. State tables separate the issue of when a function is performed from the function itself. This attribute of state tables is synergistic with the interactive state tables mechanization which lets one execute single functionally bounded modules or lists of functionally bounded modules independently. One can debug the functions and the state tables which call the functions independently. Again, it is possible to design conventional software such that the control flow portion of the algorithms call procedures to perform the computation. One can then develop the control flow code, leaving stubs for all the procedures, etc., achieving some of advantages of the NBS control system. Again RCS hides those details from the user and ensures consistency.

d. On the negative side, state tables tend to hide the sequence of execution. Explicit specification of next states and/or state transition graphs can help alleviate this shortfall.

Figure 5 shows the combination of control levels to form more complex hierachical structures. Every cycle each level executes its preprocessing, state table, and postprocessing functions. Each control level decomposes input commands it receives into output subcommands which it sends to the level below, while reporting status back to the level above. All algorithms are built using symbolic variables, which are attached to data during execution, at the point the data is required. The interface between any two levels is identical in mechanisms, hopefully providing a foundation for future modular, "plug compatible" hardware and software for robotics and other real-time sensory interactive control applications. Figure 6 shows the TASK, ELEMENTAL-MOVE (E-MOVE), and PRIMITIVE levels as they are currently implemented. The commands and data to each level and to the vision system are shown, as is the status and data back from each level.

2.2 DETAILS OF THE CONTROL LEVELS

2.2.1 TASK Level

The TASK level develops a sequence of high level goal points using object and location point names provided in its command from the workstation.

Commands from the WORKSTATION to the TASK LEVEL are:

ACQUIRE (OBJ [at A])

From its current position, the robot will go to position A, (SOURCE), and grasp the named object, (OBJ). If no location A is specified, then the object will be acquired from the robot's current location.

MOVE ([OBJ from A] to B)

The robot will acquire the named object from location A, (SOURCE), and will move it to B (DEST), but will not release the object. If (OBJ from A) is not specified, the robot will move from its current position to B.

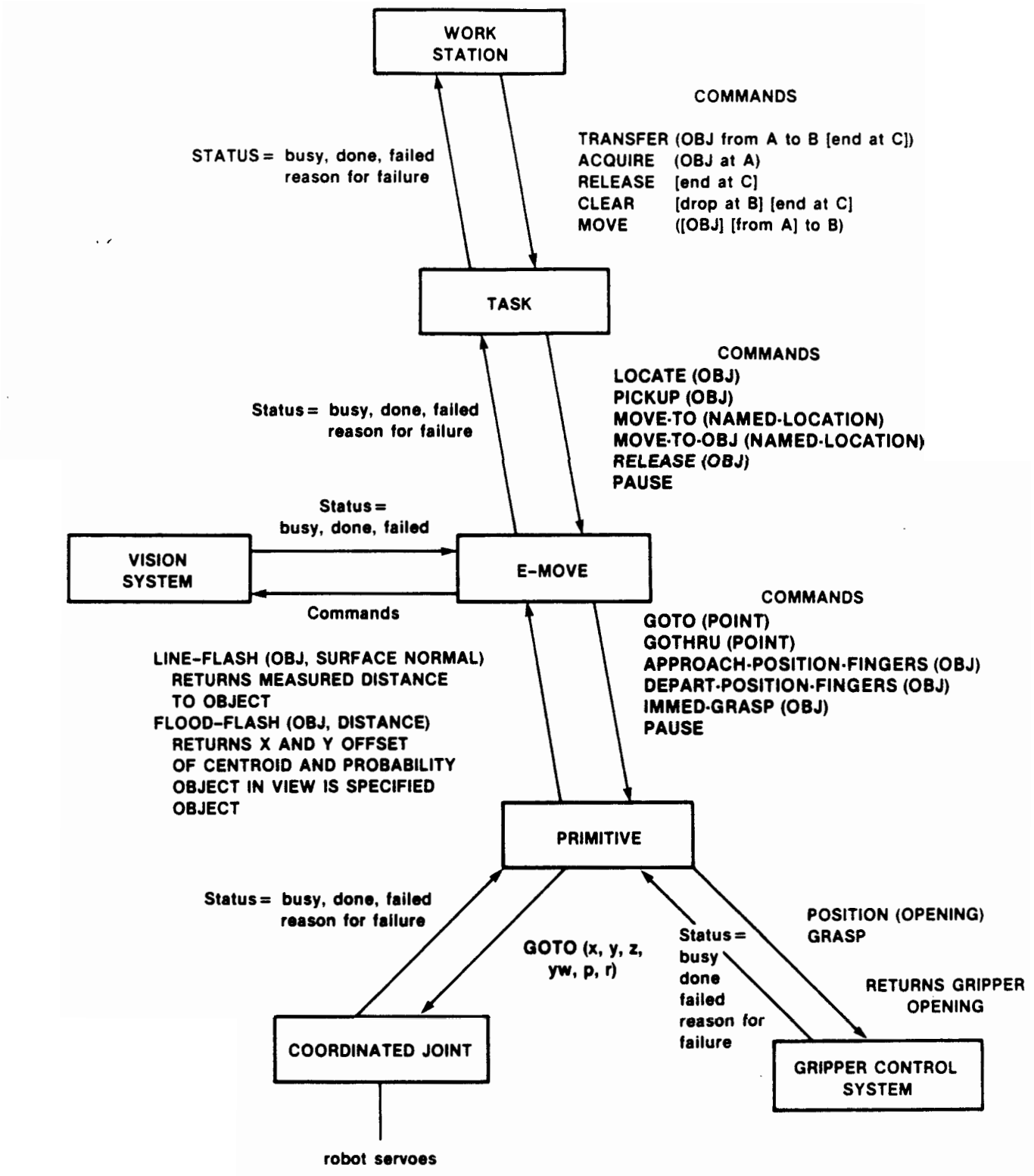


Figure 6. Commands-to Each Level of Robot Control System

RELEASE [end-at A]

The robot will release the object it is holding at its present position and move to position A. If A is not specified, the robot will move to the "SAFE" position associated with its current location (e.g., if the robot is at the fixture it will move to the fixture safe location a few centimeters above the fixture).

TRANSFER ([OBJ from A] to B [end-at C])

The robot will acquire the named object from location A, (SOURCE), release it at position B (DEST), and move to C (END-AT). If (OBJ from A) is not specified, and if the robot is already holding an object, that object will be moved to B. If (end-at C) is not specified the safe position for B will be used as the end-at location.

CLEAR (drop-at A [end-at B])

The robot will go to position A and release the part it is currently holding. Then it will end-at B. If B is unspecified the safe position of A is used.

Figure 7 shows the sequence of the above commands issued by the WORKSTATION to the TASK level of the RCS to manufacture a box, a sample part which has been manufactured automatically at NBS. The MOVE, RELEASE, ACQUIRE, and TRANSFER commands are commands from the workstation to the TASK level of the control system. The other lines in figure 7, are commands to the vise and to the machine tool controller.

2.2.2 E-MOVE Level

The TASK level issues commands to the E-MOVE level to implement the commands from the WORKSTATION. In general, the E-MOVE level develops a trajectory between the last commanded high level goal point and the current goal point. A trajectory may be simply a straight line between two goal points, or more complex, involving departure, intermediate, and approach trajectories. The commands from the TASK level to the E-MOVE level are:

LOCATE (OBJ)

First, the gripper is opened fully to allow the robot vision an unobstructed view of the scene. Then the robot will analyze the scene currently in its view. Using the vision system, it will identify the specified object and determine its orientation. The robot will then close the grippers to the approach opening for that object. Finally the robot is positioned such that closing the grippers will grasp the object. The database provides the physical characteristics of the named OBJ to the vision system. The details of the recognition algorithm are defined in the vision system. If the specified object cannot be identified, a locate failure is returned to the TASK level.

PICK-UP (OBJ)

The gripper is closed until the gripping force on the object reaches the prespecified value. The lower level will return a grasp failure if the final gripper opening is not within the required tolerance. The object size is acquired from the database and the gripper opening is part of the status data passed up from the Gripper Controller.

MAKE BOXT

COMMAND	PART	SOURCE	DEST	END AT
VISE-OPEN RIGHT, CLOSE LEFT				
MOVE	BOXT (2)	BUFFER	FIX-MAC1	
VISE-CLOSE RIGHT				
RELEASE				FIX-MAC2
MOVE			FIX-MAC3	
VISE-OPEN RIGHT				
MOVE			FIX-MAC4	
VISE-CLOSE RIGHT				
MOVE			FIX-SAFE	
ENGRAVE PART				
ACQUIRE	BOXT (2)	FIX-MAC5		
VISE-OPEN RIGHT, OPEN LEFT				
TRANSFER	BOXT (2)		V-BLK1	V-BLK2
TRANSFER	BOXT (3)	V-BLK3	V-BLK4	V-BLK2
VISE-CLOSE LEFT				
MOVE	BOXT (2)	V-BLK5	FIX-MAC1	
VISE-CLOSE RIGHT				
RELEASE				FIX-MAC2
MOVE	"CLOSE-GRIP"		FIX-MAC3	
VISE-OPEN RIGHT				
MOVE			FIX-SAFE	
MACHINE DOVE TAIL				
ACQUIRE	BOXT (2)	FIX-MAC5		
VISE-OPEN RIGHT, OPEN LEFT				
TRANSFER	BOXT (2)		STATION-x(y)	WS-SAFE

Figure 7. Commands From Workstation to Make a Box Top

MOVE-TO (LOCATION)

The robot is moved from the current location to the named destination location. The MOVE-TO command successively accesses the points along the desired trajectory, commanding the PRIMITIVE level to GOTHRU each point until the last point, at which time a GOTO command is issued to GOTO that last point.

MOVE-TO-OBJ (OBJ, NAMED-LOCATION)

MOVE-TO-OBJ is exactly the same as MOVE-TO except that the approach opening for the named object is accessed from the database and before moving, the grippers are opened to the approach opening of the specified object.

RELEASE (OBJ)

The RELEASE command opens the gripper to the release opening specified in the database for the specific OBJ at its current location.

PAUSE

Each level has a pause command. When received, it passes a "pause" down, and a done back up to the level above.

The E-MOVE level is the only level that is currently interfaced with the vision system. The vision system is capable of many complex operations including visual servoing at frame rates, and comparison of the current image with the expected image as determined from a world model. Currently, however, only two vision commands are used by the control system. They are:

FLOOD-FLASH (OBJ, EXPECTED RANGE)

The name of the OBJ and the expected range of the object are passed to the vision system. The vision system accesses the vision database to acquire the feature values of the object, and compares these features to the object or objects in its field of view. It locates the image of the correct object and returns the x and y offset in mm in camera coordinates of the object and the roll angle of the object. It also returns the range of the object based upon an assumed surface orientation.

LINE-FLASH (OBJ, EXPECTED RANGE, EXPECTED SURFACE-NORMAL-VECTOR)

Upon receiving this command, the vision system takes a 2-line structured light picture. This picture reveals the range and surface orientation of the observed object. If the measured surface orientation and range do not match with the expected surface orientation and range, then the vision system returns a "line-flash fail". Otherwise it returns the measured range. Several of the test parts manufactured in the NBS experimental factory have identical top surface dimesions and can only be distinguished by their thickness. The line flash command is used to verify that the correct part was being observed. Ranges could be resolved to .2%. This yields, for the viewing distance used, approximately 1mm. resolution. Surface orientation can be determined to approximately 2%. The vision system functioned without problem in all of the ambient light conditions experienced, and contains hardware and algorithms to help alleviate problems usually encountered in dealing with highly reflective parts.

2.2.3 PRIMITIVE Level

The PRIMITIVE level is the level which interfaces with the robot and gripper. The NBS control system fully supports having multiple systems (the robot and the gripper controller for example) functioning simultaneously. The PRIMITIVE level generates intermediate points along a trajectory defined by the E-MOVE level, and passes these points to the robot dependent control levels. The E-MOVE level also passes to the PRIMITIVE level a pointer to trajectory parameters including maximum acceleration, speed, etc.

GOTO (POINT)

The GOTO command will cause the robot to move in a straight line to the desired point. As it nears that point it decelerates and stops at the destination point.

GOTHRU (POINT)

The GOTHRU command is identical to the GOTO command except that the robot does not decelerate as it approaches the commanded point. A "done" is returned to the E-MOVE level when the robot comes within a breakpoint distance of the specified destination. The E-MOVE level then sends the next point while robot continues to move.

APPROACH-POSITION-FINGERS (END-EFFECTOR-PARA-PTR)

The system database includes an end-effector parameter table which specifies the approach opening, departure opening, and grip opening for each part. The E-MOVE level passes a pointer to this table to the PRIMITIVE level. The APPROACH-POSITION-FINGERS command passes a command and data to the gripper controller to open the gripper to the approach-opening of a specific object. If the gripper, for some reason, cannot carry out that command, a "grip failure" is reported to the E-MOVE level.

DEPARTURE-POSITION-FINGERS (END-EFFECTOR-PARA-PTR)

DEPARTURE-POSITION-FINGERS is the same as the APPROACH-POSITION-FINGERS except that the departure-opening is passed to the gripper control system instead of the approach-opening.

IMMED-GRASP (END-EFFECTOR-PARA-PTR)

This command directs the gripper controller to close the gripper until the default force of 100 lbs is encountered on the fingers. Then the part size pointed to by END-EFFECTOR-PARA-PTR is checked against the current finger spacing of the gripper. If the two values do not match within tolerance, the PRIMITIVE level returns a "grip failure" to the E-MOVE level.

PAUSE

Reports done to the level above, and directs the lower level to PAUSE.

2.2.4 Below the PRIMITIVE Level

2.2.4.1 Robot Interface

The PRIMITIVE level provides the robot nine numbers every cycle. These nine values are the x, y, z coordinates of the center of the wrist plate, defining its

position in space; the x, y, z value of a unit vector pointing normal to the wrist plate defining its orientation, and the x, y, z value of a unit vector defining the rotation of the wrist plate. This specification of the position of the robot is unambiguous, and may be sufficiently generic that it (or a similar interface) can provide the basis for a standard at some time in the future. The coordination of joint motion to achieve the required pose is robot dependent and may be performed by the robot controller, or by another control level.

2.2.4.2 Gripper Interface

Currently, only two commands are being used.

POSITION (GRIPPER-OPENING, FORCE, SPEED)

This command will servo the gripper to the specified opening at the specified speed. If the force on the fingers exceeds the specified force before the grippers are within tolerance of the specified opening, a "grip failure" is returned.

GRASP (GRIPPER-OPENING, FORCE, SPEED)

The grippers are closed at the specified speed until the force on the fingers exceeds the specified force. The gripper controller continually returns the current gripper-opening.

2.3 EXAMPLE

In order to better understand the task decomposition performed by the TASK, E-MOVE, and PRIMITIVE levels consider the command

TRANSFER FLAG FROM TRAY TO FIXTURE

passed to the TASK LEVEL. This command would be decomposed into:

MOVE-TO	TRAY
LOCATE	FLAG
PICKUP	FLAG
MOVE-TO	FIXTURE
RELEASE	FLAG
MOVE-TO	FIXTURE-SAFE

These commands would then be implemented by the E-MOVE level.

The E-MOVE level must decompose each of these commands into still finer commands. The most complex of these commands is MOVE-TO TRAY so we will describe that command's decomposition.

At some previous time, a user of RCS would have entered data into the forms system. This data would specify geometry of each part, location of each point, approach and departure paths to be followed to/from named locations, and intermediate trajectories. The E-MOVE level would successively access the departure trajectory from its current location, the intermediate trajectory, and the approach trajectory to the final goal point, issuing GOTHRU and GOTO commands to the PRIMITIVE level for each of those points. When the last GOTO has been completed by the PRIMITIVE level, the E-MOVE level has completed its task and it returns a done status to the TASK level.

INPUT	STATE TABLE ACQUIRE LINE 1	LINE 2	LINE 3	LINE 4	LINE 5	LINE 6	LINE 7	LINE 8
new-command	: (EQ) true	: (EQ) true	: (EQ) true	: (EQ) true	: (EQ) false	: (EQ) false	: (EQ) false	: (EQ) false
ref-command-in	: (EQ) pause.	: (EQ) pause.	: (EQ) pause.	: (EQ) pause.	: (EQ) move-to-obj	: (EQ) move-to	: (EQ) locate.	: (EQ) pickup
status-in	: (EQ) done	: (EQ) done	: (EQ) done	: (EQ) done	: (EQ) done	: (EQ) done	: (EQ) done	: (EQ) done
cur-state	:	:	:	:	: (EQ) 4-state	: (EQ) 2-state	: (EQ) 3-state	: (EQ) 5-state
loc-of-obj	: (NE) vis-reqd.	:	:	: (EQ) vis-reqd.	:	:	:	:
obj-acquired	: (EQ) false	: (EQ) true	: (EQ) false	: (EQ) false	:	:	:	:
source-spec.	: (EQ) true	:	: (EQ) false	: (EQ) true	:	:	:	:
	:	:	:	:	:	:	:	:
	:	:	:	:	:	:	:	:
	:	:	:	:	:	:	:	:
OUTPUT								
	: SET SOURCE	: CMD = >FAILURE	: CMD = >FAILURE	: 2-NEXTSTATE	: 5-NEXTSTATE	: 3-NEXTSTATE	: 5-NEXTSTATE	: D-NEXTSTATE
	: 4-NEXTSTATE	: F-NEXTSTATE	: F-NEXTSTATE	:	:	:	:	:
	:	:	:	:	:	:	:	:
command-out	: MOVE-TO-OBJ	: PAUSE.	: PAUSE.	: MOVE-TO	: PICKUP	: LOCATE.	: PICKUP	: PAUSE.
status-out	: EXECUTING	: FAIL	: FAIL	: EXECUTING	: EXECUTING	: EXECUTING	: EXECUTING	: DONE

INPUT	STATE TABLE ACQUIRE LINE 9	LINE 10	LINE 11	LINE 12	LINE 13	LINE 14	LINE 15	LINE 16
new-command	: (EQ) false	: (EQ) false	: (EQ) false	: (EQ) false	: (EQ) false	: (EQ) false	:	: DEFAULT:
ref-command-in	: (EQ) pause.	: (EQ) move-to	: (EQ) move-to-obj	: (EQ) locate.	: (EQ) pickup	:	:	:
status-in	: (EQ) done	: (EQ) failed	: (EQ) failed	: (EQ) failed	: (EQ) failed	:	: (EQ) executing	:
cur-state	: (EQ) d-state	:	:	:	:	: (EQ) f-state	:	:
loc-of-obj	:	:	:	:	:	:	:	:
obj-acquired	:	:	:	:	:	:	:	:
source-spec.	:	:	:	:	:	:	:	:
	:	:	:	:	:	:	:	:
	:	:	:	:	:	:	:	:
	:	:	:	:	:	:	:	:
OUTPUT								
	:	: MOVE = >FAILURE	: MOVE = >FAILURE	: LOC = >FAILURE	: PIC = >FAILURE	:	:	: STATE = >FAILURE
	:	: F-NEXTSTATE	: F-NEXTSTATE	: F-NEXTSTATE	: F-NEXTSTATE	:	:	: F-NEXTSTATE
	:	:	:	:	:	:	:	:
command-out	: PAUSE.	: PAUSE.	: PAUSE.	: PAUSE.	: PAUSE.	: PAUSE.	:	: PAUSE.
status-out	: DONE	: FAIL	: FAIL	: FAIL	: FAIL	: FAIL	: EXECUTING	: FAIL

Figure 8. State Table for ACQUIRE

The PRIMITIVE level calls an RCS function (STLINE) to calculate a straight line path between the current location and the goal passed to the PRIMITIVE level from E-MOVE. The move calculated is that to be made by the end point of the robot hand, but in addition, this level calculates the motion to be made by the robot wrist and fingers to cause a smooth transition from the present orientation to that of the goal point.

2.4 STATE TABLES

As has been described previously, the task decomposition within a control level is primarily performed through a state table. Each possible command to a control level will have a corresponding state table which will execute when that command is received.

To better understand the way in which state tables are used, we explain the ACQUIRE state table in some detail.

Looking at figure 8, the input variables in the upper left of the state table are the "table test variables" which represent feedback variables from the lower level and from sensory systems, as well as the values of internal variables. Every cycle of the control system each of these variables is compared with the test conditions in each and every line of the state table. A line is said to match the table test variable if every input value in that line matches the table test variable's value. If no lines, or more than one line matches, a STATE-TABLE error has occured. If exactly one line matches then the procedures in the output section of that line are executed. Figure 9 shows most of the procedures called by ACQUIRE. There are no restrictions as to what a procedure must or must not do. However, they will generally pass commands to the next lower levels in the hierarchy, process sensory data, pass status back to the higher levels, and set the internal machine state. In the current RCS, procedures are coded in an NBS developed language called SMACRO. However, any language would be sufficient.

We must emphasize that we have no interest at this time in defining a standard language for programming such procedures. In fact, given today's state-of-the-art, we believe any such effort would fail. Instead we are pursuing standard interfaces, and a standard structure based on hierarchical decomposition and communicating levels of the hierarchy. The language used for specific computation seems to be of secondary importance. Because the use of SMACRO is not a primary issue, we will not present the syntax or features of SMACRO. Some SMACRO procedures (with obvious meaning will be shown, but only for the purpose of explaining what is done.

The table test variables determine when commands should be executed. The table test variables for the state table ACQUIRE are:

- new command.

  True, if and only if a new command has been received from the level above. When any command is sent, a command number is incremented and sent along with the command. This incremental-command-number is used by a level to tell if a command is new. A preprocessing routine which executes each and every cycle sets the new-command flag if a new incremental-command-number is received.

- ref.-command-in

  The command received by a lower level is echoed back.

- status-in

```
COMMAND
RELATED
PROCEDURES

##  SET-SOURCE

    source-locpt-ptr-in => locpt-ptr-out
    obj-ptr-in => obj-ptr-out
    PRINT" SET-SOURCE "
    source-locpt-ptr-in => remember-current-loc
    false => apoint-pos-flag

end-routine


##  LOCATE

    obj-ptr-in => obj-ptr-out
    source-locpt-ptr-in => locpt-ptr-out
    locate => output-command
    PRINT" LOCATE "

end-routine


##  MOVE-TO
    obj-ptr-in => obj-ptr-out
    move-to => output-command
    PRINT" MOVE-TO "

end-routine


##  MOVE-TO-OBJ
    obj-ptr-in => obj-ptr-out
    move-to-o => output-command
    PRINT" MOVE-TO-OBJ "

end-routine


##  PICKUP
    obj-ptr-in => obj-ptr-out
    pickup => output-command
    PRINT" PICKUP "
    true => obj-acquired
    obj-ptr-in => obj-now-in-gripper
    obj-now-in-gripper => gripped-obj-ptr-out-to-emove
    call REMOVE-FROM-BUFFER-DB

end-routine
```

```
STATUS
RELATED
PROCEDURES

##  EXECUTING

    executing => status-report
    PRINT" executing=>status-report "

end-routine


##  DONE

    done => status-report
    PRINT" done=>status-report "

end-routine


##  PIC => FAILURE

    pic=>failure
    PRINT" PIC=>FAILURE "
    false => obj-acquired
    null-obj => obj-now-in-gripper
    null-obj => gripped-obj-ptr-out-to-emove

end-routine


##  CMD=>FAILURE

    cmd => failure
    PRINT" CMD=>FAILURE "

end-routine


##  MOVE=>FAILURE

    move => failure
    PRINT" MOVE=>FAILURE "
    null-loc. => remember-current-loc

end-routine
```

Figure 9. SMACRO Procedures Used by ACQUIRE

The status of the lower level: Executing, Done, or Failed.

- cur-state

  An internal variable used for sequencing through the states of a state table. Specific state variables are not required. Several of the state tables used in the November Run did not use a state variable for sequencing.

- loc-of-obj

  The ACQUIRE command gives the location of the object to be acquired. The database is accessed to determine if vision is required to locate the part at that location. If vision is required, table test variable loc-of-obj = vision-reqd, otherwise loc-of-obj does not equal vision-reqd.

- obj-acquired

  This flag is an internal flag which will be true if there is currently an object in the robot gripper.

- Source-spec.

  This flag will be false if the SOURCE location has not been specified.

When the ACQUIRE command is first received, table-test-variable new-command will be true, ref-command-in will be "pause", and status-in will be "done". Only lines 1, 2, 3, or 4 of state table ACQUIRE could possibly match.

Assuming the ACQUIRE command was legal, either line 1 or 4 will match depending on loc-of-obj. If loc-of-obj equals vis-reqd then vision is required. Line 4 matches, and command MOVE-TO is issued to the E-MOVE level. (Note that MOVE-TO is itself a procedure but its effect is simply to issue the MOVE-TO command.) In the case that the SOURCE location is a pallet of some type, a sector-number will be sent from the WORKSTATION as part of the ACQUIRE command, and the robot will move to the correct viewing pose for that sector. This will be addressed further in section 3.2. Procedure 2-NEXTSTATE sets cur-state to 2. While the robot is moving, the status-in value will be "executing" and line 15 will match. Eventually the robot will reach the specified goal point, the E-MOVE level will report status-in = done, and line 6 will match. The variable cur-state will be set to 3, and procedure LOCATE commands the E-MOVE level to locate the specified object using vision. LOCATE returns the x and y offsets and roll angle which will place the robot gripper directly over the object, aligned with the axis of the object. While LOCATE is executing, the TASK level state table for ACQUIRE will again match on line 15. When LOCATE completes, status-in will equal done, and line 7 will match. The procedures executed in line 7 will set the NEXTSTATE to 5, and will execute procedure PICKUP. PICKUP passes an object-pointer down to the E-MOVE level so the PICKUP command can use the database to determine the expected size of the object to be grasped and compare it with the actual finger spacing after the grasp operation; passes the command PICKUP to the E-MOVE level; sets true into object-acquired because as soon as the pickup completes, an object will be in the gripper; and sets an object pointer to indicate what is currently in the gripper.

When the E-MOVE level reports the PICKUP command is done, line 8 of the ACQUIRE state table matches and the ACQUIRE command is complete. A PAUSE is sent down to the next lower level, a DONE is reported back to the WORKSTATION, and the next state is set to the done-state. From then on, line 9 will match until a new command is received.

When ACQUIRE is a new command and the loc-of-obj is not equal to vis-reqd, then vision is not required. Line 1 will match, and the MOVE-TO-OBJ command will result in the gripper being opened to the approach-opening of the named object, and the robot moving to the source location. When the MOVE-TO-OBJ is done, line 5 matches and from there execution proceeds as already described for the case where vision is required.

Lines 10 to 13 handle the several types of failures which may occur. In these specific cases, no error recovery was attempted. The type of error is simply reported to the level above via the variable "failure" passed to the WORKSTATION.

If, when ACQUIRE was first received, either and object has already been specified or the source is not specified, lines 2 or 3 would match because the command is illegal. Procedure CMD=>FAILURE prints a message, and sets the word "command" into a variable failure" which will be returned to the WORKSTATION to indicate the type of failure which occurred. Procedure F-NEXTSTATE sets variable cur-state to a value indicating a failure. In this failure case, line 14 will match on the next cycle, commanding the lower level to PAUSE and returning a fail to the level above. Line 14 will continue to be executed until a new commmand is received.

The other state tables and procedures look and function similar to the ACQUIRE.

3. COMMON MEMORY

The communications buffers, dictionary, and database all reside in common memory.

3.1 COMMUNICATION BUFFERS

All communication between the control levels, sensory systems, robot, gripper, and other subsystems occurs via the common memory communications buffers. A convenient user interface to the communication system makes all of the implementation details of communication transparent to the user. To effect the transfer of any set of variables in any control level or subsystem to a corresponding set of variables in any other control level or subsystem the user completes the following form:

```
Preprocess name
Postprocess name
TRANSFER-FROM    LEVEL   BUFFER
TO-DESTINATION   LEVEL   BUFFER
```

The first two names, Preprocess name and Postprocess name, permit the user to specify a list of routines to execute before the data is transferred and after the data transfer is complete. The Preprocessing and Postprocessing routines can be used to handle any special handshaking protocols required. The source of the data is specified by giving the control level name and the name of the list of variables to be transferred. The destination for the data is similarly specified by giving the level and buffer names. Naturally, the type and number of data items must be identical in both source and destination buffers but the names of the data items themselves are arbitrary.

At the hardware level the communication system transfers data every cycle. Every cycle the source buffers are transferred into the common memory communication buffers. The destination buffers are updated appropriately from common memory.

Figure 10 shows the command and status buffers transferred in the current robot application of RCS.

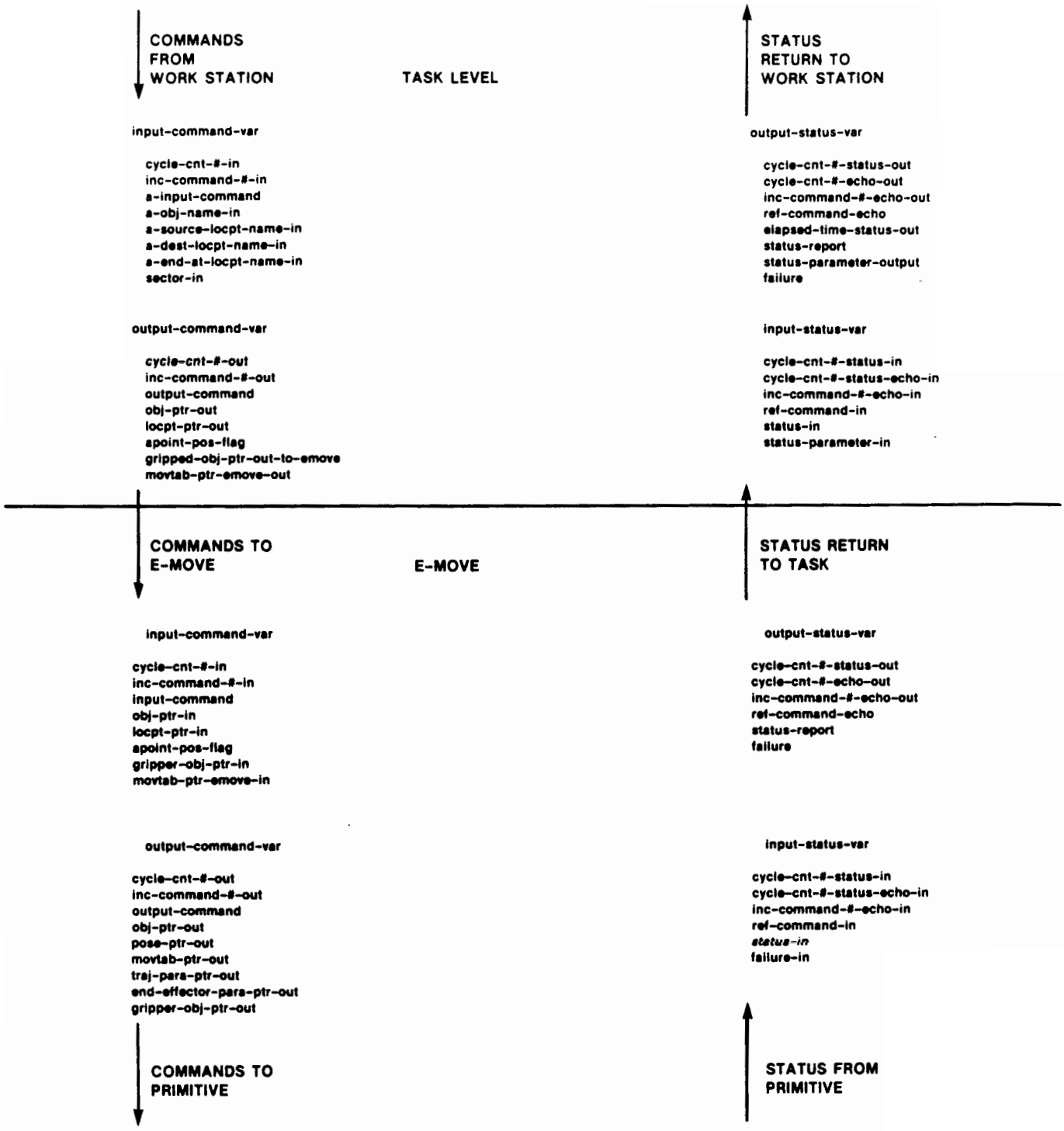


Figure 10. Communications

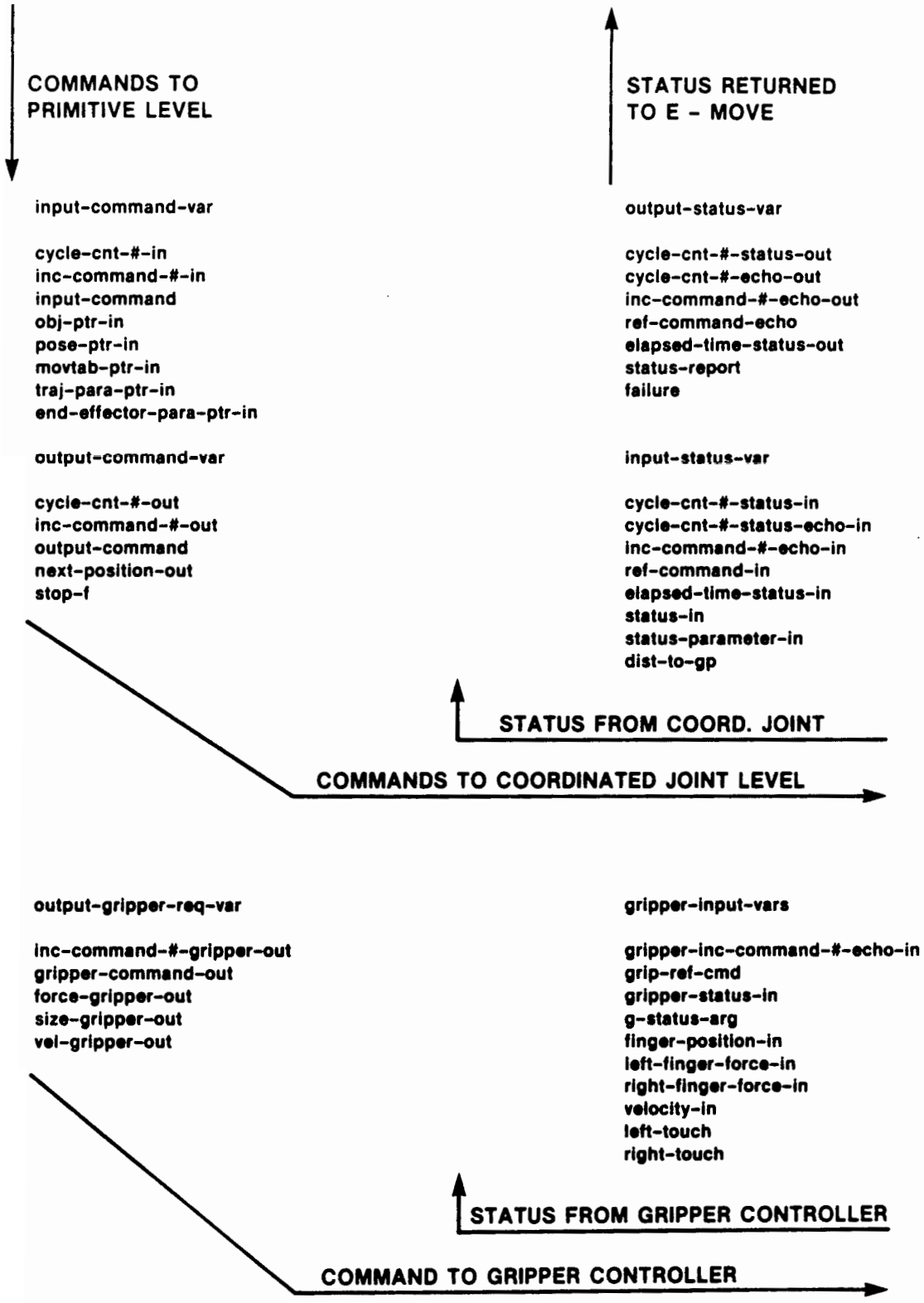


Figure 10. Primitive Level Communication (Continued)

Although the RCS Control levels execute once per cycle this does not mean that any level must necessarily respond in one cycle. From the point of view of the state tables and procedures, each level is fully asynchronous.

3.2 DATA

The control levels discussed above are only one-half of an application solution. Defining the geometry of each part, and the location of each point is also important. One of the primary objectives of the NBS Control System is to ensure that, to the maximum extent possible, commands are data driven. Changes to the part geometry or physical layout of the workspace will not change the algorithms at all -- only the data. Eventually, this data will be entered automatically from manufacturing databases containing the workstation layout, part geometry, etc.

The data which the user enters into the system define information relating to:

1. Objects, their grip size and grip location on the object, and a position from which to view each grip location.

2. Locations in the workspace such as fixtures, tables, trays, etc. Included in this data is the location of these points, and in the case of arrays of points (such as pallets) the geometry of the arrays.

3. Trajectories which define how to move from one location in the workspace to another.

All of the state tables and procedures are written using symbolic variables. Control levels attach data to these symbolic values only when the data is needed. The mechanisms which access the data structure are system procedures, almost totally transparent to users, state tables, and control level procedures.

In RCS, position and part data are entered using a Forms System. The actual data is stored in the common memory, but all of the details of storing and accessing this data are transparent to the user.

3.2.1 Poses

The most fundamental element of data required for robot programming is to associate a specific position and orientation in space space with a name.

POSE FORM

x-val	y-val	z-val	"mov-name"	"pose-name"
x_1 x_2 x_3	y_1 y_2 y_3	z_1 z_2 z_3		name
...				

The x_1, y_1, z_1, x_2, y_2, z_2, and x_3, y_3, z_3 values define the coordinates of the robot wrist plate, a unit vector defining the orientation of the wrist plate, and a unit vector defining the wrist plate rotation. Together, the coordinates of these points unambiguously define the exact position and orientation of the robot. The nine required values can be entered by moving the robot to the desired position in space and hitting the "learn" button on the joystick. In this case the nine values (x_1

to z_3) will be entered automatically. The nine values can also be entered from the keyboard if these values are known.

Column four of the POSE-FORM can be blank or can contain the name of a movtable. Movtables are used to specify offsets from any robot pose. Movtables will be discussed in more detail in the next section.

The above POSE data together determine an exact position and orientation (called a pose) in space. Every such defined pose must be given a name. This name is entered in column five of the POSE-FORM. As many poses as desired can be entered in the POSE-FORM. Each pose name represents a pose in space.

3.2.2 Movetables

It is often desirable to define positions relative to some starting pose, independent of where in space that starting pose is. The vehicle for specifying such relative offsets is a movtable and the form is a MOVTAB-FORM.

MOVTAB-FORM

w, h ho, wo r	x pit roll	y yaw	z	
type	del	del	del	"movtab-name"
w	0	0	2	
h	3	0	0	
•••				
				name

Movtables specify offsets from the current robot pose. Each line of MOVTAB-FORM specifies a specific move. There are five coordinate systems which can be used to define the offsets:

World -	x, y, z	Translate tool point along a vector defined for the work space.
Handwrist -	x, y, z	Translate tool point along a vector defined in the hand.
Hand origin - rotate	x, y	Rotate pitch (x) and yaw (y) about the current position of the fingers, leaving the tool point fixed in space.
Wrist origin -	x, y	Rotate in pitch (x) and yaw (y) leaving wrist point fixed in space.
Fingerroll -	x	Rotate fingers around the center line between the fingers.

A movtable can define any combination of up to eight translations and rotations. RCS interprets the lines in the order in which they are listed in the table. The physical motion of the robot is the resultant of all the lines of the movtable. The robot will make one smooth move from its original location to the resultant destination.

3.2.3 Objects

The user of RCS specifies the relevant features of the objects to be manipulated using three forms:

GRIP-SIZE, GRIP-LOCPT-MOVTAB FORM, and VIEW-LOCPT-MOVTAB FORM. First, we describe the GRIP-SIZE FORM.

GRIP-SIZE FORM

"object name"	ref grip #	appr. size	grasp size	grasp force	depart size
FLAG	1				
FLAG	2				
...					

Column one of GRIP-SIZE specifies the object name, and column two specifies the reference grip #. For each grip position of each object a gripper approach size, grasp size, grasp force, and depart size can be specified. If not specified, the approach size and depart size default to the gripper full open size, and the grasp force defaults to 100 lbs in the currently running system. Each named object may have many ref grip #'s. A separate line is completed for each grip position.

Form GRIP-LOCPT-MOVTAB FORM is used to specify the offsets required to define grip positions.

GRIP-LOCPT-MOVTAB FORM

"obj-name"	ref grip #	ref type	ref "loc-name"	grip "mov-name"
FLAG	1	$$$		FL-GRIP-V
FLAG	2			FL-GRIP-VR
...				

For each part, and each grip position, a movtable must be specified to define (as described in section 3.2.2) the offsets from the object origin to the desired grip orientation. It is possible that the orientation of the robot gripper may be different for different locations of the part. For example, grip position 1 for object FLAG when the FLAG is on the buffer table may require a slightly different robot gripper position than if the FLAG is in the machine tool fixture. If there is a dependence, then the grip position entry may be qualified.

The location name where the movtable specified is to be used is entered in column four ref "loc-name", and the ref type of that location (pose, location point, or array) is entered in column three ref type. The wild-card symbol $$$ can be used in column four to indicate "all locations not otherwise specified."

In the currently operating robot system, a vision system is mounted on the robot arm. Depending on the location of a part and on the particular ref grip, it may be necessary to move the robot to a view pose to observe the part with the vision system. For each ref grip of each part, at each location, a viewing position can be specified using VIEW-LOCPT-MOVTAB FORM.

VIEW-LOCPT-MOVTAB FORM

"obj-name"	ref grip	v-ref type	v-ref "loc-name"	vu-offset "mov-name"	mov-ref type	range
FLAG	1	arr	ST-1-A	GR-PIC	obj	400
...						

The object name and reference grip # are entered in the first two columns of VIEW-LOCPT-MOVTAB FORM. The expected range to the object is entered in column seven. (This has been proven useful for vision system.) As in the case of grip positions, view positions can be further qualified by specifying the name and type (columns three and four) of a specific part location. The wild-card symbol $$$ again means "every other location explicitly listed." Column five specifies the movtable which gives the offsets for the view position. For those lines of the table which specify a location, column six of the form indicates whether the movtable offsets are with respect to that location, or with respect to the object origin.

3.2.4 Location Points

A named location can be defined by specifying a pose plus a movtable. LOCATION FORM is used to specify the location and orientation associated with a given name.

LOCATION FORM

"loc-name"	"pose-name"	"movtab-name"
BUFF-SAFE	BUF-P	BUF-SAFE-M
...		

The ASCII name of the location point is listed in column one. The specification of the physical location and orientation associated with that name is entered in columns two and three. Column two is the name of a pose, and column three specifies a movtab name. That movtable gives offsets from the pose, and the resultant location and orientation is then the "meaning" of the named point.

3.2.5 Arrays

Often, a named location may not be a single point in space, but may be an array of points. A box or pallet which contains workpieces is an example. The current version of RCS permits one- or two-dimensional arrays to be given a symbolic name, just as was done for points. Whenever that symbolic name is used as a source or destination location of the robot, RCS will automatically adjust the actual position of the robot to the next full pallet position if a piece is being grasped or to the next empty position if a part is being released.

ARRAY-DIM FORM

"array-name"	# dim	# of rows	# of columns	1st "movtab"	2nd "movtab"	# pres
ST-1-A	2	3	3	y-st	x-st	9
...						

Entry "array-name" is any ASCII string which identifies the array. Column two specifies whether the array is one- or two-dimensions. Columns three and four specify how many distinct elements there are in each row, and in each column (for two-dimensional case). The geometry of the pallet is defined via columns five and six. These two entries are the names of movtables which specify the vertical and horizontal offsets for the array rows (and columns). Finally, the last entry is initialized to the number of filled locations in the pallet.

3.2.6 Owners

Sections 3.2.7 and 3.2.8 describe how to define trajectories between points. For this specification it is often useful to group a set of locations under a single owner. Trajectories can then be specified to or from any of these points with one specification to or from the owner. Without the ability to aggregate points in this manner, the number of trajectories necessary to get from any named point to any other would grow exponentially. The form for specifying owners is OWNER FORM.

OWNER FORM

"owner-name"	member type	"mem-name"
BUFFER	arr	FL-BUF-A
	arr	BB-BUF-A
	arr	BT-BUF-A
	arr	HS-BUF-A
...		

Column one is the owner name. Columns two and three give the type and name of each member.

3.2.7 Approach and Departure Trajectories

Specific objects and specific loactions will have requirements as to how the robot approaches and departs. The user must enter these values in form A/D.

A/D FORM

Appr or dept	type	"name"	ref-grip#	loc-type	"loc-name"
acc fac	vel max	smooth acc	brk-dist	ppt type	path-pt "name"
appr	arr	ST-1-A			
5	20	07	200	mtb	"name"
5	20	07	200	mtb	"name 2"
dept	obj	BOXT	3	$$$	
5	20	07	200	mtb	"name"
dept	obj	BOXT	3	lpt	V-BLOCK
5	20	07	200	mtb	VB=A/D-H
...					

The columns of form A/D have a header format and data format. The header format

Appr or dept	type	"name"	ref-grip#	loc-type	"loc-name"

identifies the trajectory. Column one identifies whether it is an approach or departure trajectory. The next two columns identify the specific beginning (for departure) or end (for approach) of the trajectory. Trajectories may begin or end at objects, location points, arrays, or owners. Hence, column two must indicate one of these four types. The third column is the name of the location point, array, object, or owners. arr, or obj. If type = obj a reference grip # is required, and the form entry may be further qualified by entering a location type (array or location point) and the name of the location. Such qualification permits the same object and grip position to be approached (or withdrawn from) differently depending on where the object is. The wild-card symbol $$$ an again be used for all otherwise unspecified locations.

The data format for form A/D is:

acc fac	vel max	smooth acc	brk-dist	ppt type	path-pt "name"

The first four entries specify parameters of motion for this part of the trajectory.

Acc-fac specifies the maximum acceleration, vel-max specifies the maximum velocity, smooth-acc specifies a smoothing parameter, and brk-dist determines how close to a goal point the end-effector must come before the robot will begin moving to the next point in a trajectory. Columns five and six specify the type and name of the specific trajectory point or offset from the previous reference pose. A trajectory points must be a pose, location point, or movtable.

As many trajectory points as desired may be listed for each trajectory.

3.2.8 Intermediate Trajectories

Approach and departure trajectories do not specify how to get from the last point of a departure trajectory to the first point of an approach trajectory. This is specified with TRAJ FORM.

TRAJ FORM

"traj-name" start-type "start name" end-type "end-name"

acc fac	vel max	smooth acc	brk-dist	type	path-pt "name"
BUF-STAT	arr	BUFFER	arr	STATION	
10	40	20	600	lpt	BUFF-SAFE
10	40	20	600	lpt	BUFF-SAFE
...					

As with the approach and departure trajectories, entries in form TRAJ have a header and a data format. The header format is:

"traj-name" start-type "start name" end-type "end-name"

These entries identify the trajectory by naming the beginning and end points of the move. Column one is a name given to this trajectory. Columns two through five are, respectively, the type and name of the start and end points of the trajectory. The only valid types for the start and end points are array, location point, or owner.

The header format for TRAJ FORM has the identical format, with the identical set of legal values as for the A/D FORM discussed previously since, in both cases, the parameters of motion for the robot, and the trajectory points themselves must be specified.

3.2.9 Data-Control Interface

RCS uses the data entered into the forms system to set up an internal structure which is then referenced by system procedures. As an example, commands SRC-CALC, DEST-CALC, and END-CALC are are called from the TASK level once for each new command. They use the object name, and source, destination, and end-at location names to access the data structures and generate the set of named goal points which are the desired path. At the E-MOVE level, command NEXT-POINT accesses each next point in the trajectory until there are no more points.

3.3 DICTIONARY

The third major part of the common memory is the dictionary. The dictionary contains records for every variable, every procedure, every state table, every list of variables, and every list of procedures. It is the dictionary which makes the interactive features of RCS possible. When a procedure name is typed on the terminal, for example, the system will look in the dictionary to locate that procedure name. Associated with that entry is a pointer which locates the actual procedure. In the above example, the procedure would then be executed.

The reason for the indirect pointer mentioned above is so that any procedure can be modified and reloaded (recompile). All references to the modified procedure point to the indirect address. When a procedure is reloaded, the pointer is updated to point to the new definition of the procedure, and all references now, indirectly, point to the new definition of the procedure. The bottom line is, therefore, that the definition of any procedure or state table can be modified, reloaded, and the entire system run with the new definition with no other actions required.

The user should interact with the dictionary only to define variables, variable owners, functionally bounded modules (procedures), executing owners, list owners, and state tables.

A user can define a variable by entering VAR "name". That variable can be initialized by entering the initial value and ! after the variable name.

A list of variables can be defined by giving the list a name, followed by its members.

```
VO "name of vo"
  VAR "name"
  VAR "name"
   •••
```

The values of every member can be seen by entering SHOW "name of VO" on the keyboard. The name of every variable will be displayed.

To define a new procedure one enters:

```
p "procedure name"
   body of procedure

   end
```

A list of procedures can be defined:

```
eo  "executing owner name"
 p   name
 p   name
 eo  name
 p   name
•••
```

As can be seen, executing owners can have other executing owners as members. When the name of an executing owner is entered on the terminal or included on the procedure side of a state table, all of the procedures in the executing owner will be executed in the order listed. This mechanism permits procedures to be combined into more complex functions while still remaining distinct, testable, and individually executable.

A list owner is a bookkeeping tool provided the user to include anything he/she likes under a symbolic name.

```
lo  "list owner name"
    name
    name
    name
    •••
```

State Tables

State tables are stored in the dictionary by calling up a blank state table form and entering the conditions and procedures. One cycle of the state table can be executed by typing the name of the state table.

4. DIAGNOSTIC SYSTEM

Because of the structure of the communication system, the relevant machine state of all processes is available in the common memory every cycle, and the diagnostic module has access to common memory data every cycle. A log of the machine state is made, permitting the NBS Control System to be stepped so the machine state at any time can be observed. This greatly facilities debugging sensory interactive real time applications where events happen at unpredictable times and there is usually no way to recreate the exact sequence of events which caused a particular problem. Diagnostic data can be printed out on a conventional terminal, transferred across a network to some central factory control computer, etc.

The Display Editor also provides an interface to graphics equipment which provides real time graphical outputs of variables. This interconnection is shown in figure 11. The Display Editor is used offline to specify what each of the connected displays is to display, where on each screen this display origin should be, and a scale factor. The real time display implemented in RCS, and the Display Editor forms are shown below.

Display type	NUMERIC
Variable	"name"
x pos	_____
y pos	_____

This display specification will cause the numeric value of the named variable to be displayed at screen position (x pos, y pos).

Display type	BAR GRAPH		
Variable	"name"	min value	_____
x pos	_____	max value	_____
y pos	_____	# pixels	_____

This display specification will cause a bar graph representing the value of the named variable to be displayed at screen position (x pos, y pos). The next two entries give the minimum and maximum expected values of the variable, and the last entry, # pixels, specifies the height (in pixels) of the display.

Display type	FUNCTION-PLOT		
variable 1	"name"	max value 1	_____
variable 2	"name"	# pixels 1	_____
x pos	_____	min value 2	_____
y pos	_____	max value 2	_____
min value 1	_____	# pixels 2	_____

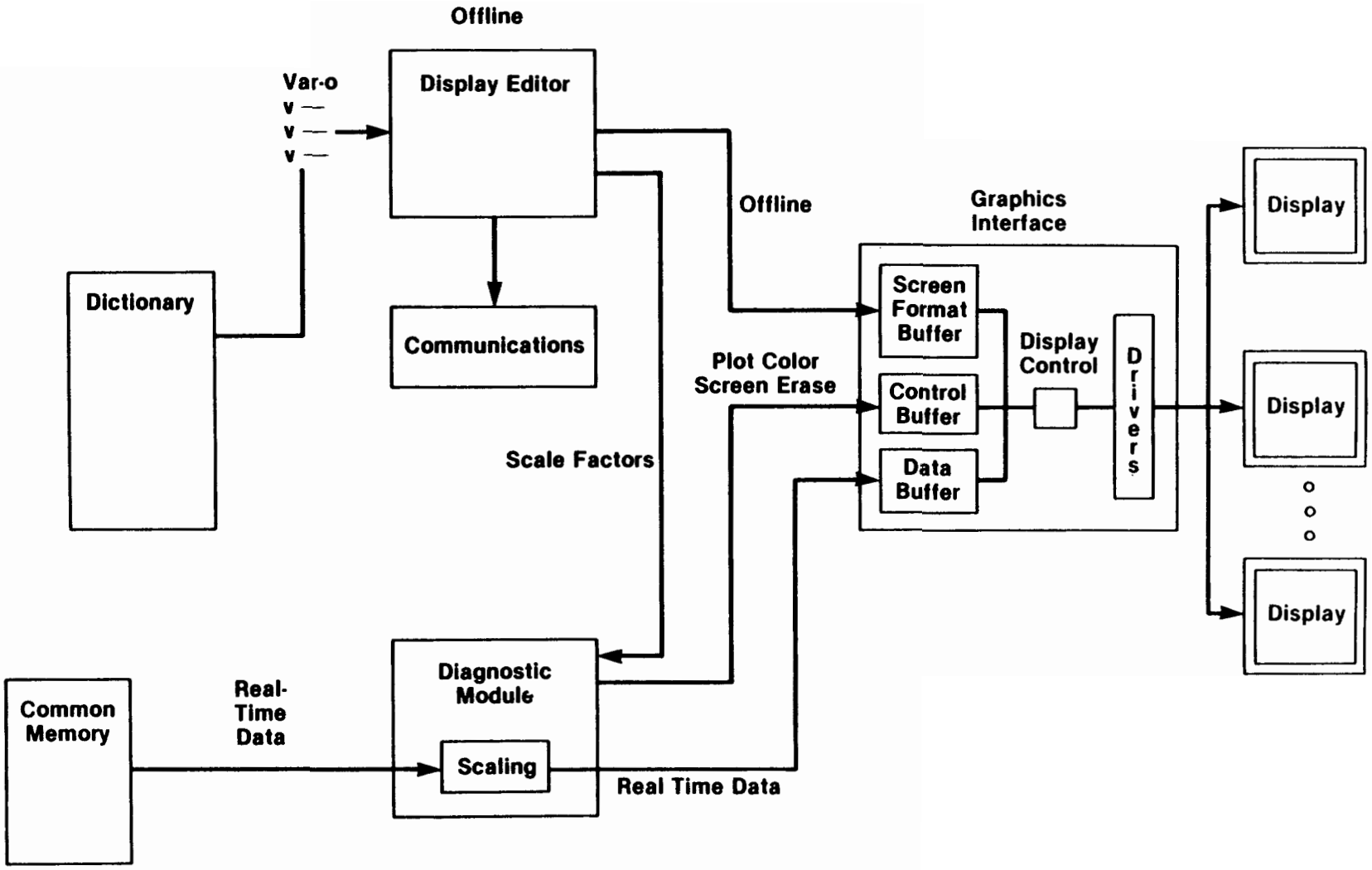


Figure 11. Diagnostic System

This display specification will cause a plot of named variable 1 against named variable 2, beginning at screen position (x pos, y pos). Min value, max value, and # pixels entries are the same as for a bar graph.

Display type	K-plot		
variable 1	_____	max value	_____
variable 2	_____	min value	_____
x pos	_____	# pixels	_____
y pos	_____		

A K-plot plots a set of vectors on the screen. The K-plot function can be used to display a stick figure robot moving on the screen, reading exactly the same data as is (or as would be if the robot is disconnected) driving the robot. To get the K-plot function to work properly, the buffers which include variable 1 must also include variable 2 and must have data in the specific order.

```
variable 1  =  x coord of vector 1
               x coord of vector 2
               x coord of vector 3
                ...

variable 2  =  y coord of vector 1
               y coord of vector 2
               y coord of vector 3
                ...
```

If the data values represent the x and y, x and z or y and z coordinates of the robot joints (from the output of the coordinated joint function) then the real-time orthographic projection of the robot arm will be displayed. Any other set of vectors can similarly be displayed. A more complex graphics system could be easily interfaced which could display a real-time isometric view of the robot.

As shown in figure 11, the offline display specification is transferred to the Screen Format Buffer in the Graphics Interface and the required communication information is passed to the Communications Module. In real time, the data is transferred from Common Memory to the Diagnostic Module and then to the Data Buffer within the Graphics Interface.

One of the additional features soon to be added to the Diagnostic System is the ability to dynamically change what is displayed based on real time data. In this case, the Diagnostic Module will test the real time data and at the appropriate time, tell the Display Editor to update the Screen Format Buffer in the Graphics Interface. All of the required format information will reside in the Screen Format Buffer and all of the real time data will be transferred each cycle. The display editor will simply enable or disable the displays.

5. FUTURE PLANS

A great deal of testing and experimentation is still needed to evaluate RCS. Based on this work, many changes and extensions will be made. Ultimately, we hope that the subsystem interfaces developed through this work will become the basis for standards leading to "plug compatible" industrial automation systems.

6. ACKNOWLEDGMENTS

Development of the NBS Real-Time Control System is partially supported by funding from the Navy Manufacturing Technology Program.

BIBLIOGRAPHY

1. Barbera, A. J., Fitzgerald, M. L., and Albus, J. S., "Concepts For Real-Time Sensory-Interactive Control System Architecture," Proceedings of the Fourteenth Southeastern Symposium On System Theory, April 1982.

2. Albus, J. S., McLean, C. R., Barbera, A. J., Fitzgerald, M. L., "Hierarchical Control for Robot in an Automated Factory," 13th ISIR/Robots 7 Symposium Proceedings, Chicago, Illinois, April 1983.

3. Albus, J. S., Barbera, A. J., and Nagel, R. N., "Theory and Practice of Hierarchical Control," Twenty-third IEEE Computer Society International Conference, Sept. 1981.

4. Albus, J. S., Barbera, A. J., Fitzgerald, M. L., "Programming A Hierarchical Robot Control System", 12th International Symposium on Industrial Robots, Paris, France, June 1982.