CAD DIRECTED AUTOMATED PART HANDLING USER'S REFERENCE MANUAL

Frederick Proctor
Peter Tanguy

September 15, 1988

This publication was prepared by United States Government employees as part of their official duties and is, therefore, a work of the U.S. Government and not subject to copyright.

Certain commercial equipment is identified in this paper to adequately describe the systems under development. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the equipment is the necessarily the best available for the purpose.

Table of Contents

I. Introduction 3
1. System Description 3
2. Purpose of This Manual 4
3. How This Manual is Organized 4
4. Who Should Use This Manual 4
5. Documentation Conventions 4

II. System Components 6
1. Hardware 6
1.1. Electrical Hardware 6
1.1.1 VAL-II Robot Controller 6
1.1.2. Unimate 2000 RCS 7
1.1.3. RCS Equipment Interface 8
1.1.4. Analog Junction Box 8
1.1.5. Force and Torque Sensor 8
1.1.6. Emergency Stop System 8
1.1.7. Washer//Dryer Interface 9
1.2. Mechanical Hardware 9
1.2.1. Unimate 2000 Robot and Controller 9
1.2.2. Unimate 2000 Gripper 9
1.2.3. Tray Stations 9
1.2.4. Rotary Vise 10
1.2.5. Washer/Dryer 10
2. Software 10
2.1. Control Structure 10
2.2. Programming Languages 10
2.2.1. VAL-II 10
2.2.2. SMACRO 14
2.3. Algorithms 17
2.3.1. `move.part` 17
2.3.2. `BOARD-LOOP` 21

III. Basic System Operation 25
1. Powerup 25
2. Errors and Recovery 26
2.1 `move.part` Errors and Recovery 26
2.2 `BOARD-LOOP` Errors and Recovery 27
3. Adding New Locations 31
4. Maintenance 33
4.1 Unimate 2000 Robot 33
4.2 VAL-II Software 33
4.3 Unimate 2000 RCS 33
4.4 Gripper and Vise Calibration 34
4.5 Washer/Dryer 35
5. Powerdown 35

Appendix A: Workstation Floor Plan 36
Appendix B: Control Schematic 37
Appendix C: Unimate 2000 Interlocks 38
Appendix D: List of References 39

I. INTRODUCTION

This manual provides a functional description of the CAD Directed Automated Part Handling (CADAPH) system of the Cleaning and Deburring Workstation (CDWS). This description includes an overview of the system, the operation of its components, and instructions for basic system operation.

1. SYSTEM DESCRIPTION

The CDWS is one of six workstations at the Automated Manufacturing Research Facility (AMRF) of the National Bureau of Standards. The CDWS cleans and deburrs the metal parts produced by the machining workstations of the AMRF. Equipment at the workstation includes a PUMA 760 Robot for deburring, a Unimate 2000 Robot for part handling and buffing, a washer/dryer system for cleaning, a rotary vise for clamping parts, and two roller tray stations for receiving parts. The workstation floor plan is shown in Appendix A.

Research at the workstation has focused on using computerized part geometry to generate accurate robot paths for both part handling and for deburring. At a computer display of a part that requires deburring, an operator selects the edges that the PUMA 760 Robot must deburr along with various deburring parameters such as tool speed and feed rate. The operator also specifies how the part is to be gripped by the Unimate 2000 Robot, how the part is to be clamped in the vise, and how the part is to be placed in the washer/dryer. These instructions, called process plans, are translated into the proper format and sent to the robots.

The machined parts enter and exit the workstation on part trays placed at the roller tray stations. The Unimate 2000 Robot, under command of the workstation controller, grasps a part as instructed by the process plan and carries the part either to the rotary vise or to the washer/dryer system. Normally the process plan specifies deburring, and the Unimate 2000 Robot puts the part in the vise where it is clamped. The PUMA 760 Robot moves to the vise and begins to deburr the edges on the top side of the part, changing its deburring tool as required. If necessary, the Unimate 2000 Robot reorients the part and the PUMA 760 Robot deburrs the edges on the newly exposed face. If a part is to be cleaned, the Unimate 2000 places it on the washer/dryer index table. From here the part cycles through the washer, where a hot water spray cleans the part, and through the dryer, where hot air jets dry the part. When all cleaning and deburring processes are complete, the Unimate 2000 Robot transfers the part back to the tray station.

To account for part misplacement in the trays, the Unimate 2000 employs the centering capabilities of the gripper and vise. A part will be centered in the closing axis of the robot gripper upon pickup from a tray. The part is then transported to the rotary vise and placed so that the closing axis of the vise is perpendicular to that of the robot. When the vise is closed, the part will be centered in this perpendicular axis also.

The inaccuracies of the Unimate 2000 are reduced with a technique known as error mapping. The response of the robot to commanded motions in the X, Y and Z axes is measured, resulting in an error map that describes how the robot will actually move when given computed coordinates. With this knowledge, part handling locations computed by the process planner can be modified so that the robot motion accurately reflects what was originally computed.

2. PURPOSE OF THIS MANUAL

The User's Reference Manual has been written to provide users of the Cleaning and Deburring Workstation with a technical reference for the part handling system. A general description of the CDWS and the part handling system has been included to familiarize first-time users with the purpose and capabilities of the system. More detailed descriptions of each component are also provided to allow an operator to troubleshoot when necessary and to modify the system if desired. Powerup and powerdown instructions for standalone system operation have been included to allow the operator to bring up and test the part handling system by itself. Powerup and powerdown instructions for the entire workstation are found in [1]. Common errors and their recovery procedures are outlined, as well as instructions for routine modifications.

3. HOW THIS MANUAL IS ORGANIZED

The manual is divided into three chapters. The Introduction gives a brief overview of the part handling system and explains the purpose and organization of the manual. The System Components chapter details the operation of the CADAPH system elements. The Basic System Operation chapter instructs the user in powerup, powerdown, maintenance, error recovery and software modification. The manual concludes with appendices that contain a floor plan of the entire workstation, a control schematic, a list of the Unimate 2000 communication interlocks, and a list of references for further information.

4. WHO SHOULD USE THIS MANUAL

This manual has been written for users who plan on running the workstation for research purposes, and intend on modifying or updating the control software or mechanical and electrical hardware. It serves as more than a description of the CADAPH system, for it contains instructions for expanding the capabilities of the workstation by adding entirely new robot locations and information for programming new applications. Users who find themselves frustrated by errors that appear during demonstrations or testing will find descriptions of common errors and their recovery, as well as hardware and software discussions that can aid in recovering from more obscure errors.

5. DOCUMENTATION CONVENTIONS

This manual uses the following conventions:

- `The name of each keyboard key mentioned in the text appears in uppercase courier letters. For example, the carriage-return key appears as RETURN.`

- In a control-key sequence, a caret (^) represents the `CONTROL` key. For example, control-C appears as `^C`, and is accomplished by holding the `CONTROL` key down, pressing `C`, and releasing the `CONTROL` key.

- Information that appears on the terminal screen appears in bold courier print. For example, the message **`*Warning* Locked in HOLD mode`** appears on the terminal when an instruction is attempted while the robot controller is in hold mode.

- Information that you must enter exactly as it appears in the manual is in plain courier print. For example, "Enter `ex move.part`" means to type `ex move.part` and press `RETURN`.

- Variable data that you must enter is enclosed with angle brackets, <>. The instruction "Enter `signal <interlock#>`" indicates that you are to replace `<interlock#>` with a specific interlock number when you enter the command. For example, you might type `signal 23` and press `RETURN` .

II. SYSTEM COMPONENTS

1. HARDWARE

1.1 Electrical Hardware

This section describes the electrical hardware used to power and control the Unimate 2000 Robot, the gripper and vise, and the washer/dryer. Only functional descriptions have been included; electrical schematics of NBS-designed equipment such as the interface boxes can be found in [2], while those for commercial products can be obtained from the vendors, which are also listed in [2].

1.1.1 VAL-II Robot Controller

Both the operating system and the robot programming language are called VAL-II, but confusions can usually be resolved through context. In this manual, the term *VAL-II controller* refers to the physical controller itself: power supply, card cage, and other items in the 19" upright rack. The term *VAL-II operating system* denotes the Unimation-written operating system software that normally runs continuously after powerup and and allows the operator to edit and run programs, execute monitor commands, teach robot locations, and save and load files. The term *VAL-II language* is used to indicate the programming language itself. The VAL-II operating system and language are described in section 2.2.1 of this chapter.

Most of the equipment located in the Unimate 2000 Robot Controller Rack is factory configured and cannot be modified, with the exception of the printed circuit boards in the card cage, which can be removed or replaced with boards compatible with the DEC LSI-11 bus. Information on such modifications can be found in [3].

The CRT in the controller rack is inconvenient for use during normal operation, since the membrane keyboard limits typing speed and the controller is placed relatively far from the robot. It is used primarily at startup, in conjunction with the control panel. When the robot has been successfully powered up the RS-232 input to the controller is switched from this CRT to one of the other terminals or the SUN computer via the 4-channel switchbox located at the terminal table.

The 5-1/4 inch floppy drive is used when programs, locations or variables are to be saved or loaded from a disk (which should be done periodically in case the battery backup fails or the system is inadvertently initialized). The power switch and baud rate switches (9600 baud) for this drive are located on the back. The drive should normally be left off, and powered on when necessary. Often the controller is reset when the floppy drive is turned on, which is harmless but annoying; simply answer NO to the two powerup messages and continue. VAL-II disk commands are explained in [3].

The control panel is used by the operator to switch power to the controller, and to select between run, hold and teach modes. Power switching is done only at powerup and powerdown (see III.1 and III.5), while mode selection is done during testing and modification. Normally, the controller is placed in run mode, while hold mode is used as a softer form of an emergency stop to pause robot motion. Teach mode is used exclusively to teach-program new locations to the robot, or to manually move the robot in joint space if it sags below joint limits during idle periods.

The power supply and controller circuitry are never used directly by the operator. The power

supply takes a 5A slow-blow fuse, which is easily replaced. Schematics for the controller are found in [4] and can be referenced if necessary.

The 24-bit interlock consists of 24 output or command relays and 24 input or status relays, and has been interfaced to the RCS Equipment Interface though a cable that runs between the Unimate 2000 Robot Controller Rack and the Unimate 2000 RCS Rack. All interfacing to these interlocks is done through the RCS Equipment Interface. The interlocks can be switched high or low (+24V or 0V) using the VAL-II `signal` command, and read using the VAL-II `signal` function (see II.2.2.1). This allows for a convenient method of digital communication between the robot- and equipment controllers.

At the bottom of the Unimate 2000 Controller Rack is the DEC LSI-11 card cage, which holds the CPU board (with batteries), the 4-channel serial board, the CMOS memory board (with batteries), analog servo boards for the robot joints, a clock board, and other boards. These boards are necessary for operation of the controller, but others may be added for more sophisticated operation, such as analog-, parallel-, or extra serial I/O boards. The serial board provided is used to read the terminal and disk drive.

The cooling fans and safety lights are located at the bottom and top of the rack, respectively. The cooling fans are turned on when the red lever at the back of the Unimate 2000 Robot is pulled up, and the safety lights are enabled whenever the robot arm power is on. Schematics for the safety lights can be found in [2].

1.1.2 Unimate 2000 RCS

The function of the Unimate 2000 Real-Time Control System (RCS) [5] is control of workstation equipment and resources. The RCS serves as a link between the Unimate 2000 Robot Controller and the equipment it relies on during the execution of `move.part`. Since the control of equipment and the analysis of its status is computation- and memory intensive, it is necessary to include an extra level of intelligence to free the robot controller for tasks for which it is better suited. The RCS can thus be thought of as a slave to the robot controller, providing the complicated sensor interfacing and error checking required during part handling and deburring.

The RCS has been implemented with an Intel iSBC 86/30 single board computer, various compatible I/O modules that allow for parallel and analog communication, a 512K common memory board, a disk and tape controller board, a Winchester disk drive for program storage, and a tape backup unit. The computer, modules, memory and I/O boards have been placed in a nine-slot Multibus backplane configured for parallel priority. All units are located in the 19" Unimate 2000 RCS Rack.

The iSBC 86/30 is an 8086-based single board computer equipped with an 8087 floating-point coprocessor. The board is configured according to the installation specifications outlined in [5]. A serial port allows for terminal interfacing, while a three-byte parallel port is used for digital I/O between the interlocks. An Intel iSBX 350 Parallel Multimodule and an iSBX 311 Analog Input Multimodule have been added for additional parallel communication to the rotary vise and tray stations and analog signal input from the Analog Junction Box. Further parallel I/O is supported with an iSBC 519 Parallel Interface Board that provides three additional three-byte parallel ports for communication with the force sensor and cleaning equipment.

The Plessey PSM-512 common memory board serves to store the SMACRO dictionary and files. The Ciprico Rimfire 45 disk and tape controller board provides low-level control over

the Priam hard disk and Cipher tape backup unit. Both the common memory board and the disk and tape controller board are configured according to the installation specifications outlined in [5].

1.1.3 RCS Equipment Interface

The RCS Equipment Interface, located in the Unimate 2000 RCS Rack, serves as a buffer and driver for digital signals between the robot controller interlocks and the RCS computer, and between the RCS computer and the equipment actuators. In the first instance, the interface serves to attenuate the 24-volt command interlock signals to the 5-volt TTL level required for input to the RCS computer, and as a driver to provide amplification of the 5-volt status information output by the RCS computer to the 24-volt level required to switch the status interlocks in the robot controller. In the second instance, the interface serves to boost the low-power output of the RCS computer to higher-power levels that drive the equipment actuators, and to ensure compatibility between the sensor feedback signals from the equipment to the TTL levels needed for RCS input. The interface also provides for the multiplexing and demultiplexing of the 24 command and status interlocks through the 8-bit RCS parallel ports, and contains a Schmitt trigger circuit used as a missing pulse detector that signals to VAL-II that the equipment control program running on the RCS computer is active. The operation of this circuit is detailed in II.2.3.2. A full schematic of the RCS Equipment Interface can be found in [2].

1.1.4 Analog Junction Box

Five analog signals returned from sensors indicate the status of various resources. Three linear potentiometers, one each in the vise, Unimate 2000 gripper and PUMA 760 gripper, and two sensors for the temperature of cleaning water and drying air comprise these signals, which are routed to the Analog Junction Box, which buffers and splits them for input to A/D converters in both the Unimate 2000 and the PUMA 760 RCS computers. In addition to its buffering and splitting responsibilities, the Analog Junction Box also supplies a precise 5-volt reference for the linear potentiometers, which require a reference voltage with minimum drift for their operation. A full schematic of the Analog Junction Box can be found in [2].

1.1.5. Force and Torque Sensor

The Lord Corporation Force and Torque Sensor measures forces and moments in three axes. The resistances of a strain gage bridge are sampled and multiplied by uncoupling and scaling matrices that produce a set of seven numbers: a status flag, three forces and three moments. The process is initiated by a start bit that is sent to the force sensor controller in the Unimate 2000 RCS Rack by the RCS system via a cable from one of the parallel boards. After a short wait, seven read cycles are signaled by the RCS system and the data are read into the parallel board for processing by the RCS software. This process is active only when signaled by the VAL-II controller and becomes idle when the interlock signal is low. A complete description of the force and torque sensor can be found in [8].

1.1.6 Emergency Stop System

To provide for safe operation of the workstation and the ability to quickly stop all activity in the event of a disaster, five emergency stop boxes have been placed at various locations around the workstation. These boxes, all wired in series, form several circuits that when broken by the appropriate switch either disable or hold equipment and robots. On each of the blue emergency stop boxes is a large red pushbutton marked "EMRG STOP" which when

pressed immediately stops both robots, the tray stations, the index table, washer, dryer and heaters. Recovering from this type of stop can only be done by pressing the emergency stop reset on the box, followed by a complete system powerup.

Less dramatic are the hold functions, one for each robot, that simply pause robot motion in mid-trajectory without disabling any equipment. When the motion is to be continued, the hold reset button on the box is pressed and motion resumes. This is useful when debugging a robot motion program for one robot without disturbing the other, and without requiring a complete system powerup after each pause.

1.1.7 Washer/Dryer Interface

The Washer/Dryer Interface, located in the Unimate 2000 RCS Rack, provides buffering and driving of input and output signals between the Acme equipment controller and the RCS computers, duplicating the function performed by the RCS Equipment Interface for the remaining equipment. It can be operated in either computer mode, where signals are sent from the computers to the equipment for control, or in manual mode, where commands are entered directly by switches to the Acme equipment. Normally, the Washer/Dryer Interface is switched to computer mode for automatic operation during robot-controlled part handling; manual mode is reserved for debugging, testing and maintenance.

1.2 Mechanical Hardware

This section describes the mechanical hardware used for part handling. Only functional descriptions have been included; mechanical drawings of NBS-designed equipment such as the rotary vise and robot grippers can be found in [2], while those for commercial products can be obtained from the vendors, which are listed in [2].

1.2.1 Unimate 2000 Robot and Controller

The Unimate 2000 Robot is a five-axis hydraulic robot used for part handling. Cables running through the trenches connect the robot to the Unimate 2000 Robot Controller Rack, which provides a CRT interface for programming and a control panel for enabling and disabling robot arm power and changing modes of operation. A Teach Pendant connected to the back of the robot acts as a joystick, allowing the robot to be manually brought to points in the workstation for teach programming. Electrical schematics for the robot and controller are found in [4]. Mechanical drawings of the robot are found in [9].

1.2.2 Unimate 2000 Gripper

The Unimate 2000 gripper is an NBS-designed split-rail parallel gripper that provides true parallel motion for both fingers. It is pneumatically actuated and fitted with a linear potentiometer for determining finger separation. Mechanical drawings of the gripper are found in [2].

1.2.3 Tray Stations

The tray stations are the shipping and receiving center for the workstations. They are controlled by the Material Handling Workstation, and take requests from the Cleaning and Deburring Workstation as input and issue grants if the AGV is not currently loading or unloading a tray at the requested station.

1.2.4 Rotary Vise

The NBS-designed rotary vise provides three-position indexing of the vise head as well as jaw opening and closing. It is pneumatically actuated and fitted with a linear potentiometer for determining jaw separation and proximity sensors to determine index orientation. Mechanical drawings of the gripper are found in [2].

1.2.5 Washer/Dryer

The cleaning system consists of an Acme washer/dryer, index table, water pump and reservoir, and air heater and fan. The index table rotates clockwise through four stops, or quadrants. The outer two quadrants are for pickup and dropoff of parts. The first inner quadrant is the washer, where hot water from the reservoir is pumped through spray jets above and below the part. The second inner quadrant is the dryer, where hot air is vented above and below the part. Valves and baffles allow for the adjustment of the volume of water and air flow, respectively. Mechanical and electrical drawings of the washer, dryer and index table are found in [6].

2. SOFTWARE

2.1 Control Structure

The control structure of the part handling workstation is shown highlighted in Appendix B. Commands are issued by the SUN WSC to the Unimate 2000 VAL-II robot controller, which represents the highest level of control in the part handling system. Robot motion instructions, variable assignment, and program instructions are executed by VAL-II directly. Equipment actuation is not directly supported by VAL-II and must be requested via the interlock signal command. Normally, VAL-II program execution is responsible for signaling equipment actuation, and the WSC has no knowledge of these actions. The WSC can, however, request equipment actuation directly in special cases. In any event, the Unimate 2000 Robot Controller must be powered on for any equipment actuation to take place from the WSC.

2.2 Programming Languages

2.2.1 VAL-II

VAL-II is the name of the computer-based control system supplied by Unimation for use with its Unimate 2000 Robot. This control system consists of the VAL-II *operating system* and the VAL-II *programming language*. The operating system provides for the creation and execution of programs, the creation of variables and robot locations, the storage and retrieval of data to and from floppy disks, and the interpretation of commands from the terminal. The programming language is an interpreted language which provides many standard programming instructions such as `if...then`, `case` and `do...until`, as well as instructions such as `move`, `approach` and `depart` that result in robot motion. In this manual, the term VAL-II can refer to both the operating system and the programming language; the distinction between the two will be clear from context. A partial description of the VAL-II operating system and language follows; however, detailed descriptions of most of the commands and instructions mentioned have been omitted for brevity. The reader is referred to [3] for a more complete discussion of VAL-II.

The VAL-II software resides in battery-backed memory and can be loaded from a floppy disk at system initialization. Once loaded, it remains intact until the system is reinitialized (rarely done), or until the batteries fail. The batteries are recharged during the period when the robot

controller is on, and will last several years before needing replacement. If VAL-II must be reloaded, follow the procedure outlined in section 4.2 of chapter 3.

VAL-II programs are created and modified with the line editor, which is invoked by entering `ed <program name>` at the terminal. Editor commands are summarized in [3]. Once a program has been created it is executed by entering `ex <program name>` at the monitor. Make sure the robot arm power is on, since programs will not normally execute with the arm power off. Programs may be halted before their normal end by entering `abort`. The current status of the robot and an executing program may be seen by entering the `stat` command. Entering `plist <program name>` will allow for a program to be listed at the monitor, while entering `dir` will result in a complete listing of all the program names currently in memory.

VAL-II includes the usual arithmetic operators `*`, `/`, `+` and `−` for multiplication, division, addition and subtraction respectively, as well as more complex functions such as `sin` and `cos`. Anywhere a number value is to appear as an argument or parameter, an expression involving operators and variables can appear instead; VAL-II will evaluate the expression and substitute the value for the argument or parameter. Logical true is represented as -1, and false as 0. If the result of an expression is to be used as a logical value, it is interpreted as true if non-zero and false otherwise.

VAL-II provides four types of variables: numeric, location, precision point and array. Numeric variables contain floating-point numbers in the range ± 6.0E-39 to ±2.8545E+38, and integers in the range -32767 to +32767 or 0 to 65535. (A numeric value is an integer if its fractional part is zero and it lies in the integer range.) The name of a numeric variable must begin with a letter, and can contain any number of letters, numbers and the `.` character.

A location variable is a description of a robot end effector location relative to its origin, and thus consists of six elements, three for Cartesian coordinates and three for Z-Y-Z Euler angles. A location variable, however, is *not* a six-element array variable, although it may be thought of as one. The name of a location variable follows the convention for that of a numeric variable. The elements of a location variable are denoted `X`, `Y`, `Z`, `O`, `A`, and `T`, the first three representing the Cartesian position and the last three the Euler orientation. The VAL-II operator `trans` returns a value of location type, which may be used anywhere a location variable may appear. The syntax of the `trans` operator is `trans(X, Y, Z, O, A, T)`, where `X`, `Y`, `Z`, `O`, `A`, and `T` are numbers or numeric expressions for the Cartesian and Euler coordinates desired. The "zero" location is `trans(0, 0, 0, 90, −90, 0)`. This represents the location and orientation of the robot origin, which is inside the body of the Unimate 2000 and is thus unattainable.

A precision point variable is a location variable whose elements are not Cartesian and Euler coordinates but values of the joint angles themselves. For the five-axis Unimate 2000 precision points have five elements, the first being the value of joint one, the second for joint two and so on. Precision points follow the naming convention for location variables, but must be prefixed with the # symbol. The VAL-II operator `#ppoint` returns a value of precision point type, which may be used anywhere a precision point variable may appear. The syntax of the `#ppoint` operator is `#ppoint(<jt1>, <jt2>, <jt3>, <jt4>, <jt5>)`, where `<jt1>` through `<jt5>` are numbers or numeric values for the joint angles desired.

An array variable is just a set of numeric, location or precision point variables that share the same name and can be addressed by indexing. The name of an array variable follows the convention for naming a numeric, location or precision point variable, with the addition of a

pair of square brackets at the end, `[]`, enclosing the index. Array indices must lie in the range 0 to 65535. A useful operator that returns an array variable is `decompose`, which takes a variable of either location or precision point type and places its elements into the elements of an array. The syntax of this operator is `decompose <array>[] = <location or precision point value>`. This allows for modification or inspection of one element of an otherwise inaccessible value. Note that the square brackets must be empty.

Variables of different types may share the same name. For example, memory may hold the following four variables at the same time with no conflict: `temp.val` (numeric), `temp.val[0]` (array), `temp.val` (location), and `#temp.val` (precision point). The choice of the proper value is decided by context.

To view the values of numeric variables currently in memory, enter `rlist`. The values of all numeric, integer and array variables will be listed in alphabetical order. To view location and precision point variables, enter `llist`. An alphabetical listing of all current robot locations will be shown.

Numeric variables can be assigned values with the VAL-II assignment operator, =. Its syntax is `<variable> = value`, where `<variable>` is any numeric variable or array element and `<value>` can be a number, numeric variable or expression. If variable assignment is to be done as a monitor command, it must be prefixed with `do`. Location variables can be assigned in two ways. The first and most common is by bringing the robot to the desired location with the teach pendant (see section V.3) and entering `here <location name>` at the monitor. The Cartesian and Euler coordinates of the current robot location are stored to `<location name>`. The second method requires using the VAL-II `set` instruction, whose syntax is `set <location name> = <location value>`, where `<location name>` is the name of the location variable that is to be assigned and `<location value>` is either an existing location variable or an expression that returns a location value such as the `trans` or `#ppoint` operator.

Location and precision point values are a necessary part of the programming language, as they provide the information required for the execution of robot motion instructions such as `move`, `approach` and `depart`. These instructions, detailed in [3], take as arguments any location or precision point value which are either stored in a location or precision point variable or calculated with the trans or #ppoint operators. As a further extension, VAL-II supports *relative transformations*, which are mathematical combinations of location values that result in the computation of one location relative to another, instead of the robot origin as is the usual case. The syntax of a relative transformation is `<location1>:<location2>`, where the `:` operator indicates that the base coordinate frame for `location2` is that defined by `location1`.

The default origin coordinate frame for the robot can be changed with the `base` command. The `base` command takes four parameters as arguments, the X, Y and Z offset of the new base relative to the default origin and the rotation about the Z-axis of the new frame. For example, executing `base 20, -30, 40, 45` will define the new robot origin to be 20 mm in the +X direction, 30 mm in the -Y direction, and 40 mm in the +Z direction of the old origin frame, rotated 45 degrees in the +Z direction. The varying position of the tool tip for differing end effectors can be accommodated with the `tool` command, which takes as its argument a location value that is considered as a final relative transformation from the unterminated end of the last joint. For example, executing `tool trans(0, 0, 250, 90, -90, 0)` will inform the robot controller that the tool tip is now 250 mm further out, but has the same

orientation as before.

The VAL-II operating system makes a distinction between *monitor commands*, or commands that are typed directly into the monitor for immediate evaluation, and *program instructions*, which are instructions normally executed during the running of a program. Monitor commands include the aforementioned `ed`, `ex`, `dir`, `plist`, `rlist` and `llist`, among others; program instructions include `if..then`, `case` and `do...until` statements as well as instructions such as `move`, `approach` and `depart` that result in robot motion. Normally, monitor commands are typed only at the monitor and program instructions are placed only in programs, but it is possible to execute a program instruction immediately at the monitor by prefixing the instruction with the command `do`. This is useful for executing robot motion instructions such as `move` without the burden of editing a one-line program, for example by entering `do move rest`. Unlike monitor commands, however, program instructions entered at the monitor require the arm power to be on, just as for executing a program.

It is often desirable to execute a program for debugging purposes with the robot either slowed significantly or stopped altogether. To accomplish the former, VAL-II provides the `speed` command, whose syntax is `speed <value>`, where `<value>` is an expression whose value lies between 0 and approximately 150. Normal operating speed is 100; smaller numbers result in slower speed while larger numbers result in faster speed. Speeds greater than about 150 do not appreciably increase the speed of robot motion. A setting of 10 slows the robot motion significantly enough so that potentially disastrous motions can be arrested before damage results. The robot arm power must of course be on.

It is possible to tell VAL-II to ignore all robot motion instructions encountered during program execution or at the monitor by enabling the dry run software switch with the command `enable dry.run`. This switch allows programs to be executed with the arm power off by directing VAL-II to ignore all subsequent robot motion commands. Typing `disable dry.run` will disable the dry run switch. To view the current settings of the software switches, simply enter `sw` at the monitor.

Digital communication with external sources is implemented with 24 command (output) and 24 status (input) interlocks, which are 24-volt relays tied to the RCS Equipment Interface. The command interlocks are numbered from 1 to 24, while the status interlocks are numbered from 1001 to 1024. The command interlocks can be pulled high (24 volts) or low (0 volts) by executing the `signal` command, which takes as arguments the interlocks numbers which are to be written. The command interlocks are made positive if they are to be pulled high and negative if they are to be pulled low. For example, the command `signal 1,-2` will drive interlock 1 high and interlock 2 low, leaving the rest of the interlocks unaffected. If digital input to VAL-II is required, the status interlocks can be read with the `signal` function, which returns the logical AND of the monitored interlocks. A high or 24-V value for an interlock is considered true, while a low or 0-V value is false. If the interlock number is made negative, this assignment is reversed. For example, `signal(1001, -1002)` will return -1 and be evaluated as true only if interlock 1001 is high and 1002 is low. Note that the `signal` *function* requires parentheses, while the `signal` *command* does not.

A program may be run asynchronously in the background by invoking it with the `pcex` command, similar to the `ex` command for programs that are run in the foreground. The "pc" stands for *process control.* Programs that are intended to be run in the background cannot contain any instructions that produce or modify robot motion, as this would conflict with robot control programs that may be running in the foreground. Instead, process control programs are intended for monitoring external signal lines, changing the values of variables, and for other

housekeeping tasks. To halt a process control program, enter `pcabort`. To check the status of any process control programs that may be executing in the background, enter `pcstat`.

Signals can be set up to be monitored continuously during the execution of a program with the `react` instruction, whose syntax is `react <signal>, <program>`. `<signal>` is the status interlock to be monitored, and `<program>` is the name of the program that is to be executed upon detection of a transition of the signal. If `<signal>` is positive, the `react` will respond to a high-to-low transition and execute `<program>`. If the signal specified is made negative, the `react` will respond to a low-to-high transition and execute `<program>`.

2.2.2 SMACRO

SMACRO is a set of macros written in FORTH and FORTH assembly language that serve as a high-level language for use with RCS. Complete descriptions of SMACRO and FORTH are provided in [5] and [7]; the pecularities of SMACRO in its implementation in the part handling system will be presented here.

SMACRO uses the same block structure as FORTH, where code is placed into sequentially numbered 1K disk blocks, and a similar though more elaborate dictionary for code bookkeeping. This compatibility between the two means that FORTH words can be imbedded in SMACRO routines and vice-versa, making the operation of SMACRO appear identical to that of FORTH. Although inextricably linked to the FORTH environment, SMACRO programs differ from those in FORTH in a number of ways. In FORTH, code is written in words, which consist of a colon, the name of the word, its definition in terms of previously defined FORTH words, and a semicolon. The colon signifies that the following text is to be compiled as a definition and not executed immediately, while the semicolon indicates the end of the definition. The first three characters and the length of the name determine its uniqueness, and the name itself can contain any characters except whitespace characters (spaces, tabs, carriage returns). Once a definition has been entered, it can be compiled with the FORTH word `LOAD` and executed by simply entering its name, allowing for simple debugging; when the word operates as desired it can then be used as part of the definition of another word. In SMACRO, code is written in routines, which begin with the word `routine`, are followed by the routine name, SMACRO commands, constructs, and previously-defined routine names, and are terminated with the word `end-routine`. SMACRO routine names can contain any character except space and carriage return and are unique for their entire length, up to 31 bytes. Like FORTH words, SMACRO routines are edited into 1K blocks using the editor (which will be described later), loaded with `LOAD`, and executed by entering the name of the routine. A routine can be used in another routine. Unlike FORTH, SMACRO code can be redefined after its inclusion in a higher-level routine without the need for recompilation of the higher-level routine; in FORTH, if a word is redefined, all other words that use the redefined word must be recompiled for the change to take effect.

The application code for part handling has been stored in disk blocks 8000 through 8999. Since `OFFSET` is usually set to 8000, these blocks will be referenced as 0 through 999. Their format follows the directory convention explained in [5], where a small number of blocks are used for the purpose of documenting the contents of the blocks that follow. The vocabulary for all SMACRO words used in the part handling application software is `Bucket2 DEFINITIONS`, or `$DEF` as an abbreviation. All blocks containing SMACRO routines must begin with this vocabulary definition.

At the bottom of each directory block is a list of information used for quick loading of all code referenced by that block. The first word in the information list is `<#> ref-blk`, which

assigns the block the relative number <#>. Following this assignment is a list of the numbers of the blocks documented by the directory block relative to <#>, followed by the word `load`. Normally, this information is commented out with parentheses along with the rest of the directory block. However, when a block contains code, its number and the following `load` may be removed from the commenting. When the directory block is subsequently loaded, all commented documentation will be ignored, but the relative loads will be executed wherever they occur. This convention allows for the rapid loading of all routines and variable declarations (and anything else that can be placed in a block) with the loading of a single block (block 3 by convention). Sample blocks are shown below.

```
3
 0 ( main load block )
 1
 2 4 LOAD      % definition of Bucket2 vocabulary is in block 4
 3 600 LOAD    % vise and acme ownership code
 4 500 LOAD    % washer/dryer code
 5 400 LOAD    % tray station code
 6 300 LOAD    % rotary vise and gripper code
 7 200 LOAD    % force sensor code
 8 100 LOAD    % BOARD-LOOP, BGO, STAT
 9
10
11
12
13
14
15
```

When the above block (block 3) is loaded with `3 LOAD`, it begins load sequence starting with block 4 and continuing through block 100. The last six blocks will themselves load much more code, as shown below for block 100.

```
100
 0 ( Bucket2 : VAL <=> Equipment
 1 ------------------------------------------------
 2  1  Variables
 3 10  STAT support code
 4 20      "
 5 30      "
 6 40
 7 50  Commands to Equipment
 8 60  Status from Equipment
 9 70  BOARD-LOOP, BGO and associated routines
10 80
11 90  STAT
12 ------------------------------------------------ )
13 0 ref-blk
14    1 load   10 load   20 load   30 load ( 40 load )
15   50 load   60 load   70 load ( 80 load ) 90 load
```

Notice that everything from the first parenthesis on line 0 through the last parenthesis on line 12 is ignored by the `100 LOAD` in block 3. Line 13 establishes block 100 as 0 for the `load` statements in lines 14 and 15. Blocks 40 and 80 (140 and 180 without reference to `0 ref-`

blk) are commented out since they contain no code.

Code is typically formatted as follows: variable declarations are made in the low-order blocks of a set of 100, while code begins at the next block of 10 after the final variable declaration. In order to compile all SMACRO routines without referencing uncompiled code, a routine that is to be used as part of the definition of another should be placed in an earlier block to allow for the quick loading feature of ref-blk and load.

A set of useful "library" routines has been written and placed in blocks 20 through 69. These include monitor control words such as dim, bright, and reverse, along with others that change the video attributes of characters on the Televideo screen. The FORTH word DC used for cursor placement has been rewritten as a SMACRO word, dc, that can be used directly in a routine without the need for the to-stack and ~F commands. This graphics set is used extensively in the STAT routine to provide appealing output. READ-A/D is also an extensively used routine that reads the analog-to-digital converter channel specified in the variable ch-#, returning the digital value in a/d-word. A quick debugging routine, A/D, can be used for realtime dump readings of the channel specified. Its syntax is <#> A/D, and produces a stream of output until the space bar is struck.

Blocks 100 through 199 contain the high-level definitions for BOARD-LOOP, the equipment control program, STAT, the realtime status display program, and BGO, the backgrounding facility. These routines are detailed in section II.2.3.2. The autoload block for the 2 D>M disk image automatically initializes the hardware ports and executes BGO and STAT upon system powerup so that no operator intervention is needed to run the equipment control programs.

Blocks 200 through 299 contain code that reads and encodes force sensor data. The operation of the code is detailed in section II.2.3.2. Of additional interest are the debugging routines in blocks 270 through 279. These routines provide realtime data dumps to the screen of all force sensor values. RAW-DUMP prints all six force and moment values and the valid-data flag. ADJ-DUMP prints these values adjusted by subtracting the first values obtained for easier viewing of changing values. HEALTH-DUMP prints the value of the health bit, which should be 2 during normal operation and 0 or -1 when the sensor is off or being initialized. These routines all cease printing when the space bar is struck.

Blocks 300 through 399 contain the vise and gripper code that actuates the rotary vise according to commands from VAL-II and returns status regularly. In addition to this code, which is compiled as part of BOARD-LOOP, the debugging routines OPEN-VISE, CLOSE-VISE, ROT-0, ROT-90 and ROT-180 actuate the vise according to their names. These routines can be entered whether or not BOARD-LOOP is running, but will only be acknowledged when neither robot is the vise owner. Another important routine, CALIBRATE, uses a least-squares algorithm to determine slopes and offsets relating the digital potentiometer readings of the vise and gripper to their openings in inches. The operation of this routine is presented in the Maintenance section.

The one hundred blocks beginning at block 400 contain code to control the tray stations. All the code is compiled as part of the definition of BOARD-LOOP; no debugging routines exist.

Washer/Dryer code has been placed beginning at block 500. Low-level routines such as FAN-ON, FAN-OFF, WTR-PUMP-ON, WTR-PUMP-OFF and the like are used by the command and status code in the high-order blocks ending at 599 to actuate equipment. These low-level routines can be entered at the terminal when BGO is not running to test and debug the

equipment, and are found in blocks 530 through 549.

Blocks 600 through 699 list the routines that implement the vise and washer/dryer lockout protocols, as well as the code for communicating between the PUMA 760 RCS and the local RCS computer. A functional description of these protocols can be found in section II.2.3.2.

Lastly, the ten blocks beginning at 980 contain calibration data for the vise and gripper potentiometers to allow for openings to be measured in inches instead of digital a/d-converter readings. These blocks must be edited when new values are measured when executing `CALIBRATE`, as explained in the Maintenance chapter.

2.3 Algorithms

2.3.1 move.part

`move.part` is the name of the VAL-II program that is invoked by the SUN WSC and causes the robot to execute part handling motions and equipment actuation. Prior to its execution, several parameters are established by the WSC during the development of the process plan, and downloaded though the RS-232 line to VAL-II as monitor commands. These parameters are numeric and location variables that take on values that are interpreted by `move.part` in assigning actual locations for the robot.

The first parameter is a numeric variable, `j`, the loop counter for program execution. This variable can take on any integer value greater than or equal to zero, and is interpreted as the number of pick-and-place cycles minus one. For example, if a robot part handling sequence consists of acquiring a part from the tray and placing it in the vise, `j` would be assigned a value of 0 since one pick-and-place cycle is required. If the part handling sequence consists of moving a part from the tray to the vise, waiting for the PUMA 760 to deburr it, then moving it from the vise to the washer, `j` would be assigned a value of 1 since two pick-and-place cycles are required.

Associated with each value of `j` are the source and destination locations for the motion cycle: `from[j]` and `goal[j]`. These array variables are chosen from this set of predefined constants: `tray11`, `tray12`, `tray13`, `tray14`, `tray21`, `tray22`, `tray23`, `tray24`, `vise0`, `vise90`, `vise180`, or `washer`, each representing a particular taught point of the same name. `move.part` will assign the appropriate taught point based on the value selected for `from[j]` and `goal[j]`. In addition to the source and destination values that result in the selection of a taught point, two offsets are needed to specify how far from the base taught points the exact pick-and-place locations are to be, since each different part requires a slightly different end effector position for pickup or release. These location variable offsets, usually small in magnitude, are `from.offset[j]` and `goal.offset[j]`.

The following example illustrates the proper syntax for setting up the parameters and invoking `move.part` to take a part from sector 3 of tray 1, place it in the vise at orientation 90, pick it up after deburring has been done at vise orientation 0, place it in the washer, retrieve it after it has been washed, and replace it in sector 3 of tray 1. This sequence is not usually entered directly by an operator, but is downloaded by the SUN WSC after the process plan has been generated and is included here for instructional purposes.

```
do j = 2
do from[0] = tray13
```

```
do goal[0] = vise90
do set from.offset[0] = trans(3.5, 17.2, -5.0, 90, -90, 0)
do set goal.offset[0] = trans(15.0, 0, -5.0, 90, -90, 0)
do from[1] = vise0
do goal[1] = washer
do set from.offset[1] = trans(0, -15.0, -5.0, 90, -90, 0)
do set goal.offset[1] = trans(0, 0, -5, 90, -90, 0)
do from[2] = washer
do goal[2] = tray13
do set from.offset[2] = trans(0, 0, -5, 90, -90, 0)
do set goal.offset[2] = trans(3.5, 17.2, -5.0, 90, -90, 0)
ex move.part
```

No extra information is included with these parameters. Because of this, `move.part` must make general assumptions about part handling. The first assumption is that each part handling sequence is composed of pick-and-place cycles, with part acquisition the result of the pick motion and part release the result of the place motion. Due to this assumption, every movement to a pick location is followed by a command for the robot to close its gripper, and every movement to a place location is followed by a command for the robot to open it. The second assumption is that robot approaches and departures are done in the world-Z direction if a part is in the gripper and in the tool-Z direction if not. This is to prevent dragging the part along the resting surface or interfering with the part as it rests. World-Z paths are specified using the `shift` operator, while tool-Z paths are specified with either the `approach` or `depart` instruction. Their syntaxes are described in [3].

Depending on the source or destination of the part, certain workstation resources may need to be acquired or actuated. A mechanism for determining this from the initial parameters alone exists in `move.part`, implemented with two numeric variables, `place` and `resource`. The `move.part` program is divided into two very similar halves, the first half for part acquisition and the second for part placement. Although small differences in each half mandate keeping them separate, much of the control is duplicated and is thus best coded with subroutines that can accept generic parameters and be called from each half of the program. To accomplish this, the numeric variables `place` and `resource` are provided with information at the beginning of each half of the cycle, which `move.part` uses when subroutines are called to move the robot.

At the beginning of `move.part`, after an initialization and checking procedure that will be explained later, the first `from[j]` value is stored into `place`, which is just the logical name of the base taught point for the pending motion. Secondly, the first `from.offset[j]` location variable is decomposed into a numeric variable array, `offset[]`, which will be used to determine the actual offset for a `move` instruction after it is scaled for known errors.

Once `place` has been assigned, it is used to determine a value for `resource`, the variable that denotes which of the four workstation resources is to be used. This is done by the subroutine `allocate.resource`. If `place` contains `tray11`, `tray12`, `tray13` or `tray14`, `resource` is assigned the value `ts1`, for tray station one. If `place` contains `tray21`, `tray22`, `tray23`, or `tray24`, `resource` is assigned the value `ts2`, for tray station two. If `place` contains the value `vise0`, `vise90` or `vise180`, `resource` is assigned the value `vise`. Finally, if `place` contains the value `washer`, then `resource` is also assigned `washer`. The physical resource represented by the value of `resource` must always be obtained before the robot moves in for part handling to avoid collisions.

Once place, offset[] and resource have been established, move.part calls the subroutine assign.loc to determine the actual six-element location variables that are to be the argument of a move instruction. By use of a case statement, the value of place is used to decide which base taught point is to be assigned to loc, one of two location variables that are returned by the subroutine. assign.loc also relies on the array offset[], whose elements were assigned values by the decompose instruction as mentioned above. These values, whose elements represent X, Y, Z, O, A, and T, are used to flag for tool-out or tool-down gripper orientations and are modified by assign.loc accordingly to map errors that are know to occur at the tool-out or tool-down orientations. Once the array elements have been modified, they are packed into the second returned location variable loc.offset via the trans function.

When moving the robot from one point to another, it is usually desirable to force it to move through a neutral "through point" to avoid unpredictable collisions. The determination of this through point is done by knowing where the robot is and where it will be sent, and assigning a safe intermediate location. Normally, at the conclusion of a robot motion, resource is saved into the variable old.resource, and when resource is assigned a new value for the upcoming motion, the two are used to calculate an appropriate through point. Unfortunately, when move.part begins executing, no previous value of resource exists to be saved (and no simple method exists for determining one), so the calculation of a through point is complicated. This is solved the following way: when move.part is first entered, old.resource is given the value no.resource. When through points are determined and this value is observed for old.resource, the through point chosen is the rest point associated with resource. Through points chosen this way sometimes result in longer trajectories, but they are guaranteed safe. Later in program execution, old.resource will have a non-null value and can be used for a more specific choice of a through point.

The choice of a through point is further complicated by the various tool orientations about the vise. For example, if the part is to be acquired in the tool-down configuration and released in the tool-out configuration (or vice-versa), a through point is required in order to avoid striking the vise corner with the part. If the before- and after orientations are the same, no through point is needed. In both cases, resource and old.resource have the value vise, so more information is needed to differentiate these situations. Since this before- and after hand orientation is contained in the downloaded offsets, these location variables are decomposed into two arrays b[] and a[] (for before and after, respectively). If the joint angles for the fifth joint differ by more than a few degrees, the appropriate through point for differing hand orientations is selected. If they are relatively close, then no through point is selected. Confusingly, the proper offset location variables for b[] and a[] depend on which half of the pick-and-place cycle is currently being undertaken. For part acquisition, b[] contains the decomposed goal.offset[j+1], while a[] contains from.offset[j]. For part release, b[] contains from.offset[], while a[] contains goal.offset[j].

The complete operation of the subroutines that determine through points is as follows: depending on which half of the pick-and-place cycle is active, either calc.from.thru.point or calc.goal.thru.point is called to set up the proper b[] and a[] arrays. Once this is completed, the subroutine calc.thru.point uses the unique sum of old.resource and resource to determine the nature of the trajectory and assign an appropriate through point. If the sum indicates a path from the vise to the vise (which occurs frequently for part refixturing) hand orientations are checked by calc.vise.thru.point to assign a through point, if necessary. If the sum indicates that old.resource has been set to no.resource, then the rest point associated with

resource is used as the through point. The location variable returned by calc.thru.point, called thru.point, is then used as the actual argument to a move instruction when robot motion is undertaken. To indicate that a through point is needed, the variable thru.stat is set to true. Otherwise, it remains false.

Although through points ensure that the robot will move along a predictably collision-free trajectory by preventing motion to potentially dangerous regions, there are many cases when it is desirable to allow the robot access to these regions to perform tasks in cooperation with another robot. Lockout protocols have been implemented in the workstation that accomplish this by allocating certain resources to only one user, who must make a request to the protocol manager and wait for a grant before entering the restricted region. The Unimate 2000 RCS system acts as this manager, taking requests from the PUMA 760, the Unimate 2000 and the SUN WSC for the vise and index table and granting permission on a first-come-first-served priority basis. This protocol is detailed in II.2.3.2.

Robot motion commences once the determination of thru.point is complete. The subroutine move.thru checks the value of thru.stat and if true executes a move thru.point instruction that moves the robot along the safe trajectory. move.thru then requests the new resource and commands it to the proper state by calling get.resource, waits for a grant, then drops the old resource by calling drop.old.resource. Both resource subroutines follow the lockout protocol. After the resource is acquired, move.part executes the instruction approach loc:loc.offset, 200, which results in the positioning of the tool tip 200 mm back in the tool-Z direction, and executes a break instruction to stop motion in preparation for the final approach to the part. If resource allocation proceeds with no waiting, the motion appears continuous until the break instruction; if the resource is in use, the robot will pause at the through point until the resource is released, at which point the approach instruction will bring it to its final position 200 mm from the part.

The final approach to the part is done carefully. Control is delegated to four subroutines: check.grip.open, which opens the gripper if it is not already open; check.resource, which makes sure the resource has arrived at the commanded state; move.loc, which slows the robot speed to 30% of normal and reads the force sensor as the robot moves to its final location, backing away if a force overload is detected; and acquire.part, which closes the gripper and opens the vise, if necessary. Upon completion of part acquisition, the robot is instructed to move up 200 mm in the world-Z direction and break its motion.

Once the robot has the part in its possession, move.part assigns to old.resource the value of resource just used, and begins the second half of the pick-and-place cycle by setting place equal to goal[j] and decomposing goal.offset[j] into the offset[] array. allocate.resource, assign.loc, calc.goal.thru.point and move.thru are then called as in the first part of the program to begin the approach to the part destination. The robot is then instructed to continue its motion to a position 200 mm above the resource, where it is paused with a break instruction in preparation for its descent with the part.

Before the part placement motion is commanded, the status of the resource is checked to be sure it has arrived at its commanded state, and if the part release is to occur at the vise, the vise is opened. move.loc is then called to bring the robot carefully to its goal location. The last subroutine called in the loop, release.part, opens the robot gripper to place the part at its destination, first closing the vise to clamp the part if placement is occurring there. As a last fixturing procedure, if the part has been placed at the vise and the loop counter j is zero, the

vise is momentarily opened then closed to insure that the part is properly seated flush on the vise surface.

Upon placement of the part, the robot departs 200 mm in the world-Z direction and breaks its motion, setting `old.resource` to the value of `resource` just used and decrementing `j`. If `j` is negative, loop is terminated, the robot is brought to a nearby rest location by the subroutine `rest`, and `reset.2000` is called to set all the downloaded parameters to zero. If `j` is nonnegative, the loop is reentered and `place` and `offset[]` are set to the following `from[j]` and `from.offset[j]` values to begin the next motion.

The initialization and checking procedure done at the beginning of `move.part`, whose discussion was postponed until now, accomplishes three things. First, a `react` instruction is executed to allow for continuous monitoring of two status interlocks: `bucket2.health` (1016) and `comp.man.sw` (1015). These interlocks should remain high but fall low when either the equipment control program running in the Unimate 2000 RCS computer unexpectedly halts or the computer/manual switch on the RCS Equipment Interface Box has been mistakenly switched to the "MANUAL" setting. The subroutines `check.bucket2.health` and `check.comp.man.sw` also check the values of these interlocks at the beginning, since a `react` instruction only monitors for a drop from high to low and won't respond if the interlocks are already low before its execution.

The second purpose of the initialization and checking procedure is to check for zero values in the downloaded parameter set. Since all values are nulled after their use in `move.part` by `reset.2000`, a zero value at the beginning of any subsequent execution indicates that the downloaded value was not acknowledged. This has disastrous results and is therefore signaled to the SUN WSC so that the parameters can be retransmitted.

Finally, the initialization and checking procedure includes the setting of `old.resource` to `no.resource` so that an appropriate safe through point can be chosen for the first robot motion.

2.3.2 BOARD-LOOP

`BOARD-LOOP` is the name of a SMACRO routine that runs continuously on the Unimate 2000 RCS system computers, accepting equipment control commands from the Unimate 2000 through its command interlocks, executing them, and returning equipment status through the status interlocks. The functional assignment of the interlocks can be found in the appendix. `BOARD-LOOP` can be executed in the background by entering `BGO` on the equipment board, and halted by entering `HALT` on the same board. Normally, `BGO` automatically executes `BOARD-LOOP` in the background when the autoload block for the `2 D>M` disk image is loaded at system powerup, so no direct typing of `BGO` is required. If `BOARD-LOOP` has been halted, however, it will be necessary to start it again before `move.part` is executed or an error will result.

The program is divided into two parts, the first responsible for receiving and executing commands from the RCS Equipment Interface and the second for determining and sending status back to the interface. Recall that the interface acts as a buffer and driver for signals sent between the RCS computer's parallel port and the VAL-II interlocks, since the RCS computers use a 0 to 5 volt TTL range for signals while the robot controller uses a 0 to 24 volt range. The interface may therefore be considered as a simple amplifier /attenuator that ensures that logic levels will be compatible between the two systems, and its operation is transparent. With this

in mind, the purpose of the first half of BOARD-LOOP is to read the command interlocks, decode and execute the commands, while the purpose of the second half is to read equipment status, encode it and return it to the status interlocks, all communication taking place through the parallel port. This parallel port is memory-mapped starting at address 0C8 hexadecimal, and consists of three eight-bit words that are configured for reading or writing in positive logic. The lowest address, 0C8, is configured for writing, and it is though this port that the 24 status bits for the interlocks are multiplexed. The next address, 0CA, is configured for reading; the 24 command bits are multiplexed through this port. The next address, 0CC, is configured for lower-half output and upper-half input. This port is used as the select line for multiplexing. The last address, 0CE, is the control word that can be written to for initializing the previous ports.

Detailed descriptions of all the routines that make up BOARD-LOOP will not be included in this manual, except when necessary to clarify unusual operations or algorithms. Therefore, when a routine is referenced in this manual, it is advisable for the reader to locate the routine online with the LOC function, and read through it to verify its operation.

In order to keep cycle times constant, BOARD-LOOP calls two routines, START-TIMER and WAIT-TIMER, to load a clock timer in the 86/30 board Programmable Interval Timer chip with an initial count value at the start of the program cycle, and wait until the counter decrements to zero at the termination of the main code before starting the next cycle. The timer is decremented synchronously with the on-board clock, thus providing a program-independent realtime counter. The cycle time has been adjusted to approximately 47 milliseconds.

The execution of commands is first begun by EXECUTE-BUCKET2-CMD, which in turn can be broken down into three operations: the initial reading of the command interlocks, the decoding of the bit patterns into their representative equipment commands, and the subsequent execution of these commands. The routine INPUT-BUCKET2-CMD accomplishes the first by writing the select word for interlocks 1 to 8 to the select port, reading the input port and placing the word into bucket2-lo-in-cmd, writing the select word for interlocks 9 to 16 to the select port, reading the input port and placing the word into bucket2-mid-in-cmd, and finally writing the select word for interlocks 17 to 24 to the select port, reading the input port and placing the word into bucket2-high-in-cmd. INPUT-BUCKET2-CMD also calls READ-A2 and READ-C2 to input PUMA 760 vise commands and lockout requests.

Execution continues with DECODE-BUCKET2-IN-CMD, which calls routines to decode commands for the tray stations, rotary vise, force sensor and washer/dryer. DECODE-L/U-CMD simply masks off all but the tray station request bits from bucket2-high-in-cmd and puts the result in l/u-in-cmd, while DECODE-FORCE-IN-CMD masks off all but the sensor enable bit from bucket2-high-in-cmd, putting the result in force-in-cmd. The operation of the other two routines, DECODE-ROT-VISE-IN-CMD and DECODE-ACME-IN-CMD, is complicated by the fact that lockout requests and commands come from several sources.

DECODE-ROT-VISE-IN-CMD first extracts the vise request bits from the Unimate 2000 and the PUMA 760, and calls ASSIGN-VISE-OWNER to determine which robot is to receive vise privileges. Based on this determination, either DECODE-2000-ROT-VISE-IN-CMD or DECODE-760-ROT-VISE-IN-CMD masks off all but the vise command bits from either the Unimate 2000 or the PUMA 760, and places the result in rot-vise-in-cmd.

DECODE-ACME-IN-CMD performs a similar function, extracting request bits from both robots and decoding commands for washer/dryer and index table actuation from the SUN WSC. ASSIGN-ACME-OWNER will assign the washer/dryer volume to the SUN if indexing is required, and will only allot the volume to a requesting robot when indexing is complete.

After the input and decoding procedure that establishes the values for the input command variables and common-volume ownership, EXECUTE-BUCKET2-CMD calls three routines that execute the commands represented by the variables. The first is EXECUTE-L/U-CMD, which simply writes the two bits that request or release each tray station to memory address 072 hexadecimal, the address of one byte of a parallel port tied via the RCS Equipment Interface to the Material Handing controller underneath the tray stations. The second is EXECUTE-ROT-VISE-CMD, which checks for a valid command and if found writes it to memory address 070 hexadecimal, which is tied to the Equipment Interface that drives the pneumatic valves at the rotary vise. The last is EXECUTE-ACME-CMD, which looks for transitions of the variables washer-cmd, dryer-cmd and index-cmd and calls the appropriate routines that write commands to the memory-mapped parallel port wired to the Washer/Dryer Interface. The details of these routines, OUTPUT-WASHER-CMD, OUTPUT-DRYER-CMD, and OUTPUT-INDEX-CMD, are best understood by following the code listings in the appendix or online with the LOC function.

The second half of BOARD-LOOP begins with the routine EXECUTE-BUCKET2-STAT, which reads status from all devices, encodes it, and sends the information to the Unimate 2000 status interlocks via the RCS Equipment Interface. Digital status, such as the rotary vise orientation, index quadrant, and tray station owner, is provided continuously to the parallel ports by the RCS Equipment Interface and can be read as binary data. Analog status must be actively requested by BOARD-LOOP, such as vise or gripper jaw openings from potentiometer readings or water temperature from temperature sensors; this is accomplished with the READ-A/D routine. Force sensor data, digital data that is read in packets of seven, must also be actively requested according to a parallel data transmission protocol.

The tasks accomplished by EXECUTE-BUCKET2-STAT are decomposed into several subtasks and assigned to lower-level routines, similar to the hierarchy in EXECUTE-BUCKET2-CMD. At the start of each status cycle, CLR-STAT is called to clear out all the status variables from the previous cycle. Four routines are then called to input equipment status, set the corresponding status variables, and check for errors. EXECUTE-ROT-VISE-STAT inputs the vise orientation and jaw opening, and returns their values, also comparing them with the commanded values and signaling an error if they are not equal after a certain time delay. Vise orientation is determined with proximity sensors and presented by the RCS Equipment Interface to a parallel port as digital data; vise opening is determined with a linear potentiometer and presented by the Analog Junction Box to the A/D converter as analog data. EXECUTE-L/U-STAT and EXECUTE-ACME-STAT operate similarly, inputting status through either parallel ports or the A/D converter, returning the actual status and the timeout comparison to the commanded values for error checking.

EXECUTE-FORCE-STAT operates somewhat differently: unlike the previous three routines, which return status every cycle, the force data is returned only when command interlock 24 is maintained high by VAL-II; when interlock 24 is low, no force checking is done. Force data consist of forces and moments in the X, Y and Z axes and a health byte which indicates proper sensor operation, and are input according to a parallel data transmission protocol outlined in [8] in which values are sent to a parallel port when initiated with a start bit and acknowledged after every received value. When the data are first requested by VAL-II, the initial readings are stored as an offset which is subtracted from every subsequent set of readings to null the effects

of sensor and gripper weight, resulting in data that reflect the forces seen at the tool tip. These forces are compared to a predefined threshold and an error is signaled if any of the values exceed this threshold. This error signal is used by VAL-II to react immediately to force overloads that may result from misplaced parts. The health byte is also returned to VAL-II through the status interlocks to insure that valid data are being received.

There are two other signals sent to VAL-II during the status procedure: a health bit that signifies that BOARD-LOOP is running, and the computer/manual switch setting of the RCS Equipment Interface. The BOARD-LOOP health bit must be implemented in hardware, since it is a software check: if the computer is reset, no software can signal this condition. A Schmitt trigger circuit is used as a missing-pulse detector which remains high as long as the multiplexing select signals from BOARD-LOOP are received by the RCS Equipment Interface; when the pulse train ceases the output of the Schmitt trigger falls low. This output is hard-wired to the 24-volt driver for status interlock 1016. VAL-II uses this interlock to react immediately when BOARD-LOOP fails, pausing move.part execution until BOARD-LOOP is resumed. The computer/manual switch setting is hard-wired to an RCS input parallel port and is read by BOARD-LOOP every cycle. When this switch is set to "manual", the bit in the parallel port is grounded and an error is signaled to status interlock 1015, which causes move.part to pause execution until the switch is returned to "computer" mode.

Once the various statuses have been input and checked, they are formatted into bytes that are multiplexed through the eight-bit parallel port and sent to the RCS Equipment Interface, which demultiplexes the data and outputs it to the VAL-II status interlocks.

III. BASIC SYSTEM OPERATION

1. POWERUP

1) Between base of Unimate 2000 Robot and green/glass partition, lift red-handled circuit breaker to "ON" position to connect power to Unimate 2000 Robot Controller.

2) At air valves between load/unload tables and base of Unimate 2000 Robot, make sure both air valves are turned counter-clockwise to supply air to Unimate 2000 Robot gripper and load/unload tables.

3) At air valve next to rotary vise, make sure air valve is turned counter-clockwise to supply air to rotary vise.

4) At Unimate 2000 Robot Controller TV950 terminal, turn switch on black switch box next to terminal to "2000 CRT" mode.

5) At Unimate 2000 Robot Controller Rack:

   a) On front of rack, flip power switch to "ON" position to engage fan and CRT.
   b) Open front door to rack. The words **UNIM 25B 4-23-82** will appear on the terminal screen and the orange "ATTENTION" light will be on.
   c) Pull the red stop button out if necessary and push in button marked "CONSOLE POWER" to connect power to controller console and green button light will come on. Also, the orange/red "HOLD/ERROR" light will come on and the following will be displayed on the terminal screen:

```
* VAL-II BOOT B2K6.2.0 19JUL85 *

LOAD VAL-II FROM FLOPPY (Y/N)?
```

   d) Enter N. The following will appear on the screen:

```
VAL-II 2670.2.0 P/N-935F25 H/W-0 S/N-0 20MAR85 23KW
COPYRIGHT 1981 - UNIMATION, INC.
INITIALIZE (Y/N)?
```

   e) Enter N. The prompt "**.**" will then be displayed on the screen. The orange "HOLD" light will go off, but the red "ERROR" light will remain on.

6) At RCS Rack for Unimate 2000 Robot:

   a) On front of rack, flip power switch to "ON" position to connect power to rack.
   b) Open front door to rack. On top side and behind the front panel of Bucket2 interface box, flip switch to "COMP" mode to enable computer interface to equipment.

7) At RCS Terminal for Unimate 2000 Robot:

   a) Flip power switch on back of terminal to "ON" position.
   b) On reset box on top of terminal, press momentarily the reset button labelled "System Reset" to reset RCS's computer system. The words **pF86-1.4 R45-1.2 up** will appear on the screen.

c) Enter `HEX F7 C9 OUTPUT 26F0 30 ERASE 0 CBOOT 0 MBOOT init-cm 2 D>M`. If while entering this command string you make a mistake, use the `DEL` key to backspace and then re-enter the command correctly. After a short time, the system will initialize itself and the workstation status will appear on the screen. The RCS program is now running.

8) At RCS Rack for Unimate 2000 Robot, open front door to rack and look at front panel of Lord F/T Sensor Controller box. If green light marked "SENSOR HEALTH" is off and "FAILURE ALARM" is sounding, press button marked "INITIALIZE" momentarily to reset controller. Make sure the lower toggle switch of the Acme Interface Box has been turned to the "ON" position, then switch the upper toggle switch to the "ON" position to engage the relays.

9) At the Acme Power Panel behind the RCS racks, pull the red and black power lever up to the "ON" position. At the Acme Pushbutton Panel by the buffing wheels, pull the lit red "STOP" button out, and push the green "CONTROL POWER ON" button in. Power will now be supplied to the Acme equipment.

10) At Unimate 2000 Robot Controller Rack:

a) *** Important: Clear Unimate 2000 Robot's work volume of all personnel. The maximum reach of robot is delineated on the floor using safety tape.
b) Turn switch marked "ARM POWER" to "ON" position to enable power to robot and corresponding orange light will come on. After a short time, the red "ERROR" light will go out. Also, the red safety light towards the end of the Unimate 2000 Robot's boom will come on.

11) At Unimate 2000 Robot Controller TV950 terminal:

a) Flip power switch at back of terminal to "ON" position.
b) Turn switch on black switch box next to terminal to "TV950" mode.
c) Enter `EX POWER.UP`. Place your hand over the red EMRG STOP button in case of error. The gripper should open and the robot should raise slightly, move to its rest position near the trays, and execute its warm-up motions for ten cycles.
d) At the conclusion of the power-up routine, the robot should return to its rest position at the vise, and the message **`The 2000 has been reset`** should appear on the screen.
e) To run part handling system under control of workstation controller, turn switch on black switch box next to terminal to "SUN" mode. System is now fully running and integrated.

2. ERRORS AND RECOVERY

2.1 `move.part` ERRORS AND RECOVERY

There are twelve routines that may be called by `move.part` in the event of an error during its execution. `error.assign` and `error.data` check for consistencies in the data downloaded by the SUN WSC and operate locally to `move.part`, while the remainder of the error routines (all prefixed with `error`) key off of status interlocks signaled by `BOARD-LOOP`.

`error.assign` will cause a program halt and the printing of the message **`Assignment error`** at the terminal upon the detection of an invalid `from[j]` or `goal[j]` parameter

when that parameter is attempted to be used to calculate an actual robot location during `assign.loc`. This error will only occur if a mistake in the downloading procedure is undetected by the SUN WSC and is passed to VAL-II. The downloading procedure must be repeated and `move.part` reexecuted to recover from an assignment error.

At the beginning of `move.part`, two `react` instructions set up continuous monitoring of status signal lines 1015 and 1016, the first which drops low when the computer/manual switch in the RCS Equipment Interface is set to "MANUAL" and the second which drops low when `BOARD-LOOP` stops running. When interlock 1015 drops low, the routine `error.comp.man.sw` prints the message **`Comp/man error`** at the terminal and pauses the program until the computer/manual switch is reset to "COMPUTER". The program then resumes. When interlock 1016 drops low, the routine `error.bucket2` prints the message **`Bucket2 error`** at the terminal and pauses the program until `BOARD-LOOP` has been started again. In both cases, the program resumes when the corrective action is taken.

The routine `check.prog.data`, called at the beginning of `move.part`, checks for zero values in the downloaded data to ensure that no needed data has been omitted. If a zero value is found, `error.data` is called to print the message **`Data error`** at the terminal and halt program execution. The downloading procedure must be repeated and `move.part` reexecuted to recover from a data error.

Two errors from the force sensor may be reported, one if the force sensor controller is turned off or malfunctioning when force data is requested, and one if the magnitude of force sensor values is greater than the operating threshold. In the first case, the message **`Force sensor error`** is printed at the terminal and the program is halted; in the second the message **`Force exceeded`** is printed and the robot is instructed to move 200 mm in the World +Z direction, insuring that no damage results from a force overload. In both cases `move.part` must be restarted after the cause of the error has been removed. This may mean turning the force sensor controller on, initializing it, or checking cabling; or removing a misplaced part that caused the force overload.

Two errors from the robot gripper may be reported, one if the gripper closes completely indicating a missed part, and one if the gripper jaws do not close within a specified amount of time indicating a malfunctioning gripper. A message, **`No part in gripper`** or **`Gripper timeout,`** is printed if one of these errors is detected, and the program is halted. The first error can be corrected by ensuring that the part is placed properly before pickup, or that the robot location for pickup was the proper location for that portion of the program. If a gripper timeout is reported, it usually signifies that air pressure has not been supplied to the gripper and that the valve at the base of the robot must be opened. However, since the vise opening is sensed with a linear potentiometer, a number of problems may manifest themselves in one of the above errors. The Analog Junction Box may be turned off, or the cable from the linear potentiometer may be broken, or the A/D converter in the RCS computer may be operating incorrectly. The cause of the error must be removed before `move.part` can be executed again.

`BOARD-LOOP` can report three errors to VAL-II from the rotary vise: a timeout if the commanded orientation is not achieved within a preset duration, a timeout if the commanded vise opening is not attained within a preset duration, and an error if the vise closes completely indicating a missed part. All three errors halt the execution of move.part, and report the appropriate **`Rotary vise error`**, **`Vise timeout`**, or **`No part in vise`** message. These errors can be corrected in much the same way as they are corrected for the gripper, by checking for proper air pressure and part placement, or by tracing the linear potentiometer

through the Analog Junction Box to the A/D module in the RCS computer.

The last error routine, `error.l.u`, is called by `check.resource` and will halt `move.part` and return an error if the proper tray station was not locked in prior to a motion to the trays. `error.l.u` prints the message **`Load/unload error`** at the terminal; this error usually means that the tray station was not requested by the SUN WSC before the Unimate 2000 was given part handling instructions. The part handling sequence must be restarted after the proper tray station has been requested for successful operation to occur.

Other problems may alias themselves as one of the above errors. The most insidious is an inadvertent change in the tool or base transformation via a typing or syntax error. A change in either the tool or base transformation will alter the absolute position of all robot location variables and will cause unpredictable and usually fatal actions. The tool and base transformations are initialized by the program `reset.2000`, or with the commands `tool hand.tool` and `base 0,0,0,0`.

2.2 BOARD-LOOP ERRORS AND RECOVERY

When the Unimate 2000 RCS is first booted, the autoload block for the 2 D>M disk image (block 1421 absolute) restores the SMACRO definitions that make up `BOARD-LOOP`, initializes the 8087 chip, loads the function keys, then calls the SMACRO words `INIT` and `BGO`. `INIT` initializes all the communication ports and clears the status variables for fresh operation, while `BGO` places `BOARD-LOOP` in the background for execution to allow for development and debugging while the program is running. This procedure is done automatically whenever the system is booted with `2 D>M` during powerup, and normally no intervention need take place for full operation of the equipment control code relied upon by `move.part`.

A facility has been provided for realtime viewing of the status of the workstation at the terminal. Called `STAT`, this SMACRO routine displays a two-column table at the terminal and fills in the values of the status variables both local and passed to VAL-II through the interlocks. Since `BOARD-LOOP` is placed in the background, `STAT` can be entered at the terminal after powerup has taken place without interfering with its operation; the display (which is updated every half-second) can be terminated by simply hitting the space bar. A carat moves across the bottom of the screen to indicate when `STAT` is active. A motionless carat indicates that the RCS computer or terminal has ceased functioning.

The first column contains vise, tray station, force sensor and gripper data, and also includes the status of `BOARD-LOOP` and the computer/manual switch in the RCS Equipment Interface. The second column contains the status of the washer, dryer and index table. The following sections detail each entry of the `STAT` program columns and what actions should be taken in the event errors occur.

The **`vise owner`** entry shows which robot has control over the vise volume according to the vise lockout protocol implemented in `ASSIGN-VISE-OWNER`: the **`760`**, the **`2000`**, or **`none`**. `BOARD-LOOP` will only take vise commands from the vise owner, who also has the privilege of entering the vise volume. If a non-owner requests the vise, the request is queued and granted when the vise becomes available; when the vise changes ownership the source of commands switches to that new owner. A fourth **`invalid`** possibility for the **`vise owner`** entry is possible. If this entry is observed it signifies that the `ASSIGN-VISE-OWNER` routine is malfunctioning, and the error should be traced through it. **`invalid`** is used in most of the status reports to signify that one of the variables being monitored cannot be classified, and is

only attributable to software failures, which are rare (and have never been observed since the initial debugging of STAT).

The **vise position** entry displays the current jaw opening, either **open** or **closed**. An **invalid** entry indicates that PROCESS-VISE-STAT is not working properly and should be checked.

The **vise orientation** entry will contain either **0**, **90**, **180**, or **transit**, depending upon the current position of the vise as seen by the proximity sensors. When the vise is positioned at one of the three stations, the associated entry will be displayed; when it is not at any station then it is assumed to be in transit. An **invalid** entry indicates that PROCESS-ROT-STAT is not assigning the proper value to the variable rot-in-stat, and the operation of the code should be traced to isolate the error.

The **status** entry normally contains the message **ok**. If the commanded vise position was not attained after a suitable time interval, however, the message **timed out** followed by a blinking star is printed. Recovery from a timeout error is discussed in the previous section under **error.vise.time**. Whenever a blinking star error occurs, the reverse-video message **type ^R ^E to reset errors** appears at the lower left of the screen; once recovery has been complete it is necessary to type ^R ^E (for Reset Errors) to clear the error from the status screen. The ^R ^E procedure is necessary after every error accompanied by a blinking star, since these types of errors are latched in software and will not automatically disappear with the cause of the problem. The **status** entry may also be **no part in vise** printed in reverse video, which occurs when the vise closes completely. The recovery from this error can be found in the previous section under **error.vise.part**. **invalid** is the last possibility for this entry, and indicates a software failure of ENCODE-ROT-VISE-STAT.

The next entry to the first column is **L/U stations**, and contains information pertinent to the tray stations. If a request has been made to the tray station controller that has not been granted, the **waiting...** message will be printed. This message will be replaced with the appropriate **station 1**, **station 2** or **both** when the AGV releases its control of the tray stations. **invalid** indicates the failure of the routine ENCODE-L/U-STAT.

Below this entry are two messages, **force sensor** and **force limit**, that relay information regarding the force sensor. These two messages are coupled in the following way: if the force sensor is commanded to be off, the two (in order) read **off** and **---**. If the force sensor is commanded to be on and the force readings are within the threshold, the two read **on** and **ok**. These two message sets indicate normal operation. If the force is exceeded, the two read **---** and **exceeded**, followed by a blinking star and the **^R ^E** message (see above). The robot will automatically pull away when a force overload is reported; recovery usually means removing a misplaced part struck by the robot. If the force sensor is not functioning (the controller has been turned off or has not been initialized, or the cabling has been disconnected), the messages **dead** and **---** will appear, along with the blinking star. Once the sensor problem has been corrected by powering up and initializing the controller and checking cabling, type ^R ^E to remove the error message from the screen.

The setting of the computer/manual switch in the RCS Equipment Interface is printed to the screen under **comp/man sw**, and can be either **computer** or **manual**. Normal operation produces the **computer** message, while inadvertently leaving the switch on "MANUAL" results in the latter message, a blinking star and **^R ^E**. In this case, after the switch has been

placed back to "COMPUTER", type ^R ^E to reset the error.

A special variable, bgo-check, is set to 1 at the beginning of each BOARD-LOOP cycle and is checked at the beginning of each STAT cycle, where it is set to 0 afterward. If this variable is found to be 0 when checked by STAT again, it signifies that BOARD-LOOP failed to set it to 1 and therefore is not running. When this error is detected, the message **dead** in reverse video is printed after the label **bgo**. STAT should then be exited by typing the space bar and BGO should be entered again to place BOARD-LOOP in the background. When STAT is again run, this message should change to **running**.

The last entry in the first column is **2000 gripper**, which is set under normal circumstances to either **open** or **closed**. If the reverse-video message **No part in gripper** is observed, it means that the gripper is closed completely and has either missed the part, or that the robot controller has not yet been turned on to allow for the gripper to open. The first error is fatal and requires human intervention to place the part in its proper location for pickup. The second situation is often observed when the workstation is being powered up, and will resolve itself when the powerup procedures have been completed. Finally, the **2000 gripper** entry may be **invalid**, which like all **invalid** entries in STAT means that a software failure has occurred, in this case with PROCESS-GRIPPER-STAT.

Status for the washer/dryer and index table has been placed in the second column. The first three lines contain washer information, and are labeled **washer**, **water level**, and **water temp**. The **washer** entry can be **off**, indicating that the water pump has been commanded to be off and is indeed not pumping, or **on**, indicating that the pump has been commanded to be on and is pumping. The third possibility for washer is **timed out** (with a flashing star), which means that the commanded action has not been attained after a suitable time interval. This error will be observed when the pump "runs on" after being requested to stop, or does not turn on after being requested to do so. If the **timed out** error is observed, the power to the Acme equipment has most likely not been turned on. However, this could signal a problem in the water pump itself, or in the connections between the Pushbutton Panel and the RCS Interface Box or the Interface Box and the RCS parallel port. Remember to type ^R ^E to clear this latched error after removing the cause of the problem.

water level and **water temp** should both be **ok**. If the water level is below the teflon proximity switch, however, the reading will be **lo**. Water should be run into the tank to bring it to its proper level. If the water temperature is not above the minimum water temperature (about 150˚ F), the water temp entry will be **lo**. Since the water heater must be manually turned on with the WTR-HTR-ON command at the RCS terminal (or with the switches on the front of the Washer/Dryer Interface in manual mode), this normally indicates that the operator has not turned the heater on, or that not enough time has elapsed to bring the eighty gallons of water to operating temperature. However, a value of **lo** may mean that the heating coils are not receiving current, and that power to the coils should be traced back to the Power Panel or control signals from the Washer/Dryer Interface should be traced up to the Power Panel. The schematics for the Washer/Dryer Interface can be obtained from Robot Systems Division for this purpose; see the note in the Appendix regarding this.

Dryer status appears just below that for the washer. There are two entries, **dryer** and **air temp**. The first operates exactly like the washer entry described above, with **off**, **on** or **timed out** (with a flashing star) as possibilities. If **timed out** is observed, similar error-tracing should be initiated. An **ok** reading for **air temp** indicates that either the dryer has not been commanded to be on and therefore the temperature is not relevant, or that the air

temperature has reached its operating level after a warmup interval when the fan is blowing. A **lo** reading means that the coils are not producing heat at the proper rate, and that power and control signals to the coils and controller should be traced similar to the water heater coil procedure. Both the water and air heater coils have maximum-temperature shutoffs provided by Acme, which turn off power to the coils in the event that temperatures above 200˚ F are observed.

The last two entries in the status table are **index table owner** and **index quad**. The first shows the current owner of the washer/dryer volume as allocated by the lockout protocol implemented in ASSIGN-ACME-OWNER. If neither robot has the volume and index table rotation is not pending, this entry will be **none**. If one of the two robots has been granted permission to enter the volume, the entry will be the appropriate **2000** or **760**. This information is passed to the controllers of both robots and the indexing software so that one robot will not attempt to enter the volume when it does not have permission, or the table will not be indexed while a robot is working there. If index table rotation is pending or underway, the entry will be **sun**. This is to ensure that neither robot attempts to place or retrieve a part while the table is moving. Once the table has arrived at its final commanded quadrant, the SUN ownership is dropped and given to a requesting robot, if any. As with most entries in the status table, **invalid** signifies a software failure, in this case that the ASSIGN-ACME-OWNER routine has incorrectly assigned a value to acme-owner.

index quad provides the proximity-sensor view of the current state of the index table, either **one**, **two**, **three**, **four**, or **off**. Each of the first entries is observed when the numbered bin aligns with quadrant one, which is the exit quadrant just after the dryer. (Bins rotate through the fixed quadrants). **one** indicates that bin one is above quadrant one, **two** indicates that bin two is above quadrant one, and so forth. A value of **off** means that either the table is indexing and no bins align, or that power has not yet been supplied to the Acme equipment.

3. ADDING NEW LOCATIONS

The purpose of CAD-directed automated part handling is to remove the need for teaching part-specific locations to the robot by computing the Cartesian and Euler components of the locations from a knowledge of the part's geometry and placement and the coordinate frame of the robot. Although CAD data allows for the computation of robot locations that would otherwise have to be taught, the inaccuracies of the robot over its large work volume mandate teaching a small set of base points which partition the volume into smaller regions over which robot inaccuracies are less dramatic. Robot locations are then specified with the VAL-II relative transformation operator using an appropriate base taught point as the first argument, followed by a computed offset that is much smaller in magnitude. This method takes advantage of the repeatability of the robot over its taught points while benefiting from the flexibility of CAD-computed robot locations. It is necessary, however, that the base taught points represent the locations of fixed resources in the work volume of the robot, and are therefore constants that will be taught once at the installation of the robot and only updated for changes in the configuration of the workstation.

There are currently eleven taught points in the repertoire of the Unimate 2000: four at each of the tray stations, six at the vise, and one at the entrance quadrant of the washer/dryer. Two imaginary lines have been drawn down the centers of each tray station, one lengthwise and the other widthwise, to partition it into quadrants; points have been taught 5 mm above the centers of these quadrants with the gripper oriented so that it opens parallel to the front of the tray. There are two taught points for each of the three vise orientations, one point chosen for the tool-down configuration and one for the tool-out configuration. The taught point at the washer

has been placed at the center of quadrant two, flush with the matted surface of the grill.

Other less important taught points are also used. Through points and rest points, while not required to be taught with any degree of accuracy, are necessary for safe operation of the `move.part` program. Debugging often requires the definition of taught points, as does the execution of warmup procedures that smooth out the action of the hydraulics. Since these points are not used for accurate part placement or for relative transformations, they can be easily modified with little or no effect on the operation of `move.part`. A listing of all taught points currently stored by VAL-II can be seen by entering the `llist` command.

It is anticipated that these taught points may have to be "moved" periodically due to upgrades in the equipment or modifications in their arrangement, and that entirely new points may need to be defined in the event that new equipment is added. The procedure for this is simple, and requires that the robot first be placed in teach mode with the arm power on by turning the mode select switch to "TEACH" and the robot arm power switch to "ON". Both switches are located at the control panel in the robot controller.

It is now possible to move the robot to the desired location to be recorded with the teach pendant, which is an orange keypad connected with a spiral cable to the back of the robot near the control and power cable connections. The teach pendant has membrane-keyboard buttons that are used to select the mode of operation and the direction of robot travel in the selected mode. The mode select buttons are labeled "WORLD", "TOOL", and "JOINT", and when depressed select the coordinate frame in which the robot motion will be undertaken. Below these three are six other buttons, labeled "X/JT1", "Y/JT2", "Z/JT3", "RX/JT4", "RY/JT5", and "RZ/JT6", that are used to drive the robot in the direction desired. These six lower keys are further marked with plus and minus signs so that motion in both the positive and negative direction of the selected coordinate frame may be accomplished. In World or Tool mode, the first three of these produce translations of the tool tip in the World or Tool X, Y or Z directions, while the last three produce rotations (plus and minus directions governed by the right-hand-rule) about the World or Tool X, Y or Z directions. The definitions of the coordinate frames for World and Tool mode are given in [9]. Notice that when World or Tool mode is selected, the tool tip will translate in straight lines, which typically means that all joints must move in concert. In Joint mode, each key moves only its associated joint; no other joints will move. The definition of joint numbering is given in [9].

The speed of the robot during teach motion can be varied using the speed toggle switch on the left side of the pendant. Three speeds can be selected. When very precise positioning is required, the "TICK" button may be pressed, which results in a fine but very slow positioning. When this feature is selected, the direction button must be repeatedly pressed and released, each time resulting in a "tick" of the tool tip in that direction.

When the robot has been brought to the desired location using one or more of the modes discussed above, the values of its Cartesian and Euler components or its joint angles can be stored to a location or precision point variable with the VAL-II `here` command. The syntax of this command is `here <name>` or `here <#name>`, and when entered at the monitor will result in the current values being displayed on the screen along with the query **`Change?`**, where they can be modified before being stored to the variable `<name>`. If no changes are desired a carriage return will store the unmodified taught values.

When the teaching procedure has been completed, return the controller to run mode by turning the mode select switch to "RUN". This will allow for normal execution of robot programs.

The VAL-II command `point <name>` or `point <#name>` will print the current values of

the specified location or precision point variable <name> and bring up the same **Change?** message, which allows for convenient modification to taught points without the need for moving the robot with the teach pendant.

4. MAINTENANCE

4.1 Unimate 2000 Robot

The Unimate 2000 requires periodic lubrication and hydraulic fluid replacement, adjustment of the servovalves with a nulling pendant, and optical encoder calibration and lamp replacement. These procedures are detailed in [9]. A servovalve nulling pendant functionally equivalent to the pendant supplied by Unimation has been constructed and placed in the robot controller cabinet. This pendant can be used to drive each joint individually by simulating the analog voltage signals to the servovalves normally supplied by VAL-II. Instructions for the use of this pendant can also be found in [9].

4.2 VAL-II Software

If the VAL-II software is suspect, reload it by turning the disk drive on and placing the VAL-II floppy inside, answering Y to the **Load VAL-II from floppy? (Y/N)** message, and responding with 4007 when asked for the robot serial number. Wait several minutes until the system queries with **Initialize? (Y/N)** and answer Y if you want a data-free memory or N if you want to retain the programs, locations, variables and current system settings.

After each programming session, save the programs, locations and variables with the store <filename> command, where <filename> is by workstation convention all<MMDDYY>.v2. In the event that the VAL-II memory has been reset, you can reload the last save with the load all<MMDDYY>.v2. Disk commands are summarized in [3].

4.3 Unimate 2000 RCS

RCS supports disk storage of images of RAM with the PRESERVE and MEM>DISK commands. These images can be reloaded to RAM with the RESTORE and D>M commands. Using these commands significantly reduces the time that would otherwise be taken to load all software routines with the LOAD command. Five images of on-board RAM and nine images of common memory can be saved, the former with <#> MEM>DISK and the latter with <#> PRESERVE, where <#> is either 1 through 5 or 1 through 9, respectively. These commands are normally executed after all routines have been loaded with LOAD commands and tested for proper operation. Whenever the system is powered up, these images can be loaded quickly by entering <#> RESTORE <#> D>M, with the appropriate digits substituted for <#>. The user is responsible for documenting the applications associated with each PRESERVE and MEM>DISK by commenting the blocks listed with PMAP (for PRESERVEs) and DM? (for D>Ms).

Whenever an image is brought into on-board RAM with the <#> D>M command, a block associated with <#> is automatically loaded. This facilitates startup procedures by automatically executing SMACRO words typically entered after powerup (such as commands to run application code). These "autoload blocks" can be listed with the <#> AUTO? command, and edited to contain whatever is desired.

Customization of the system is also supported with the <#> CUSTOM command. This command loads a block associated with <#> that normally contains "goodies" such as the prompt string, default terminal type for use with the editor, and other special features specific to each user. <#> is an even number, usually 0; other numbers are used if multiple computer boards are to be added in the future. The user is responsible for editing the custom block, which can be listed with the <#> CUSTOM? command.

By convention, the images associated with 1 PRESERVE 1 MEM>DISK contain the base RCS only (no user application software). This image can be created by powering up RCS with HEX F7 C9 OUTPUT 26F0 30 ERASE 0 CBOOT 0 MBOOT init-cm and entering RCS. After a lengthy period during which all system definitions for SMACRO are loaded, entering 0 CUSTOM will customize RCS to the likes of the user. 1 PRESERVE 1 MEM>DISK completes the creation of the image. Whenever new application routines have been debugged, this image is brought in with 1 RESTORE 1 D>M, the routines loaded, and <#> PRESERVE <#> MEM>DISK entered. The <#> for each should be unused by other images to avoid overwriting images that already exist. To bring in the new application software after a powerup, simply type <#> RESTORE <#> D>M. The base RCS code and the application routines will all be brought in to memory.

It is important to periodically save copies of all blocks containing routines to a tape backup. If the 0 CUSTOM block is set up according to [5], entering 5 D>M will bring in an image containing several SMACRO words used for making tape backups. After placing a tape into the tape drive and pushing the "LOAD" button, entering TREWIND will cause the tape to rewind to the beginning of the first file. Entering <file#> TGOTO will position the tape at the start of the sequential <file#>, which is usually the next unused file as documented on the front of the tape. <startblock#> <endblock#> ADD-TO-TAPE will copy the range of blocks to the current file. The tape can then be rewound and unloaded with TUNLOAD, then removed from the tape drive by pressing the "UNLOAD" button. The contents of the new file should be written on the label of the tape. When a copy from tape is to be placed back into memory, the procedure is duplicated, except that ADD-TO-TAPE is replaced with FROM-TAPE.

4.4 Gripper and Vise Calibration

A set of sized aluminum calibration blocks have been machined for the calibration of the linear potentiometers for the gripper and vise. The six blocks, located in the cabinet at the workstation, vary in size from 1/2 in. x 6 in. to 3 in. x 3 1/2 in. Their edges are numbered from 0 to 11, in ascending size order.

A simple routine has been developed to calibrate both the vise and gripper pots simultaneously. It is first necessary to type ex cal.gripper at the VAL-II terminal with the arm power on, which results in the positioning of the robot just above the vise and the monitoring of status interlock 1005. Entering 900 LOAD at the RCS terminal will begin the calibration routine CALIBRATE, which queries the user to place successive block edges parallel to the vise jaws and type the space bar when completed. Both the vise and gripper will close on the part, and the pot readings will be recorded. The vise and gripper are then opened, and the next block is requested. When the procedure is complete, messages will be typed to the RCS screen indicating proper values for the slopes and offsets as determined by a least-squares algorithm, and reminders to return the robot to its rest position. These slopes and offsets are stored to their associated SMACRO variables automatically; it is the responsibility of the user to edit these values into block 980 (see section 2.2.2) so that they can be easily reestablished upon the next system powerup.

4.5 Washer/Dryer

The Acme Washer/Dryer system requires periodic lubrication and air filter changing. These procedures are detailed in [6]. The water tank must periodically be drained when it is determined that the water has fouled. A spigot has been placed at the back of the tank underneath the wooden floor supports. Hoses are located nearby for draining. The water may be drained directly out the back door onto the parking lot *only if no anti-fouling fluid has been added to the water*. If anti-fouling fluid has been added, the water must be disposed of according to EPA guidelines available from the NIST Environmental Safety Office. It is suggested that only pure water be used in the tank. This requires more frequent draining but is easier to maintain and much safer. Water can be refilled from the sink in the Shops Auxiliary Room via the same hose used to drain the tank.

5. POWERDOWN

1) At Unimate 2000 Robot Controller TV950 terminal, Enter `DO MOVE REST.VISE.` Place your hand over the red EMRG STOP button in case of possible malfunction or operator error. If robot was resting at the load/unload tables, the robot will swing to assigned rest point. If robot was resting at the vise, the robot will stay at assigned rest point. Robot is now at power down position.

2) At Unimate 2000 Robot Controller Rack, turn switch marked "ARM POWER" to "OFF" position to disable power to robot and corresponding orange light will go out. Also, the red safety light towards the end of the Unimate 2000 Robot's boom will go out. Push the red stop button to turn off the controller, shut the rack, and turn the rack switch to the "OFF" position to disable the fan and CRT.

3) At RCS Terminal for Unimate 2000 Robot:

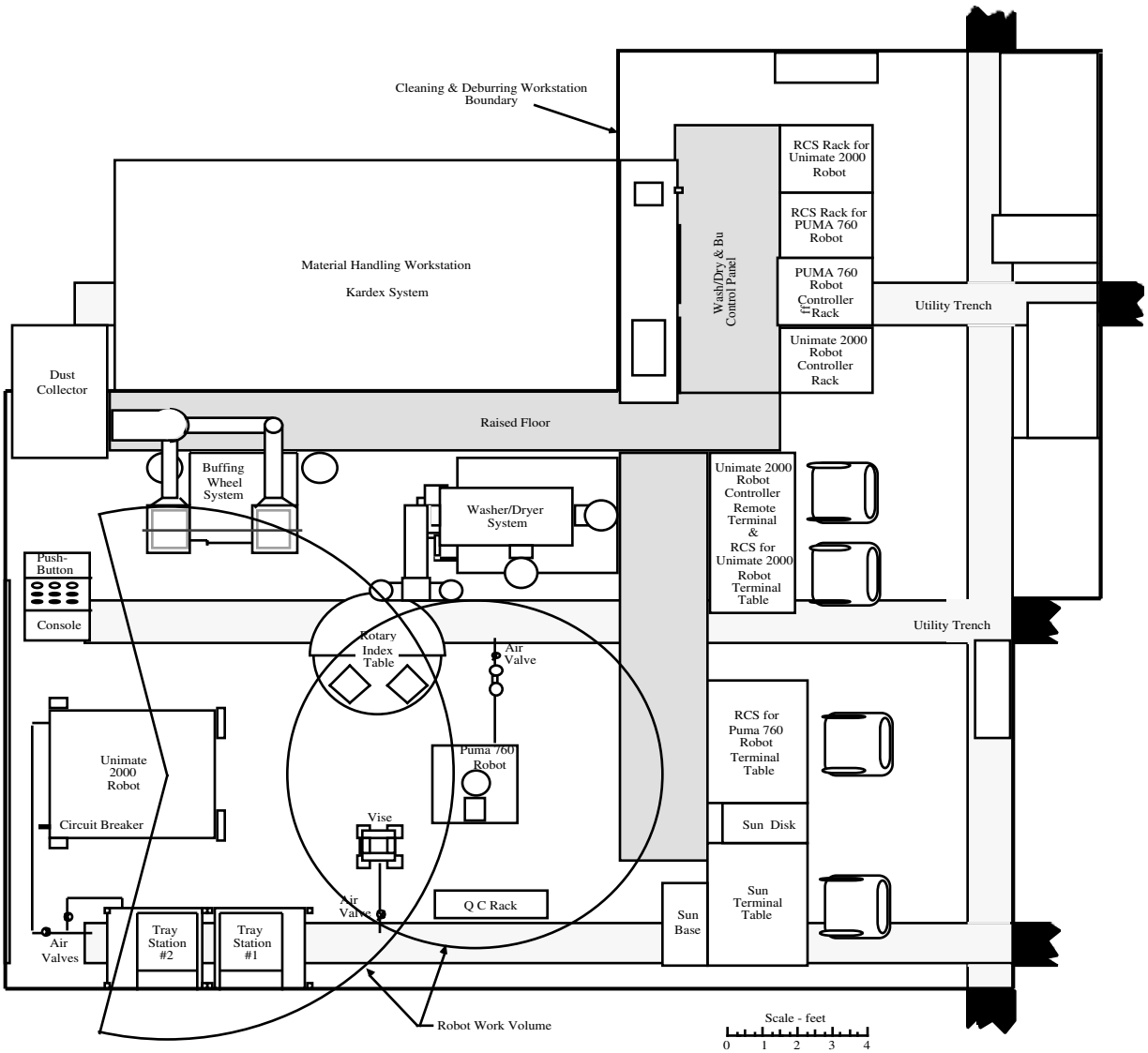
   a) Press the space bar to abort the status program. Enter `HALT`.
   b) Enter `BYE` to shutdown the RCS's storage disk. Blinking stars should fill the screen, indicating the system has been halted.
   c) On back of terminal, flip power switch to "OFF" position.

4) At RCS Rack for Unimate 2000 Robot, turn the top toggle switch of the Acme Interface Box to the "OFF" position, then turn the power switch on front of rack to "OFF" position to disconnect power to rack.

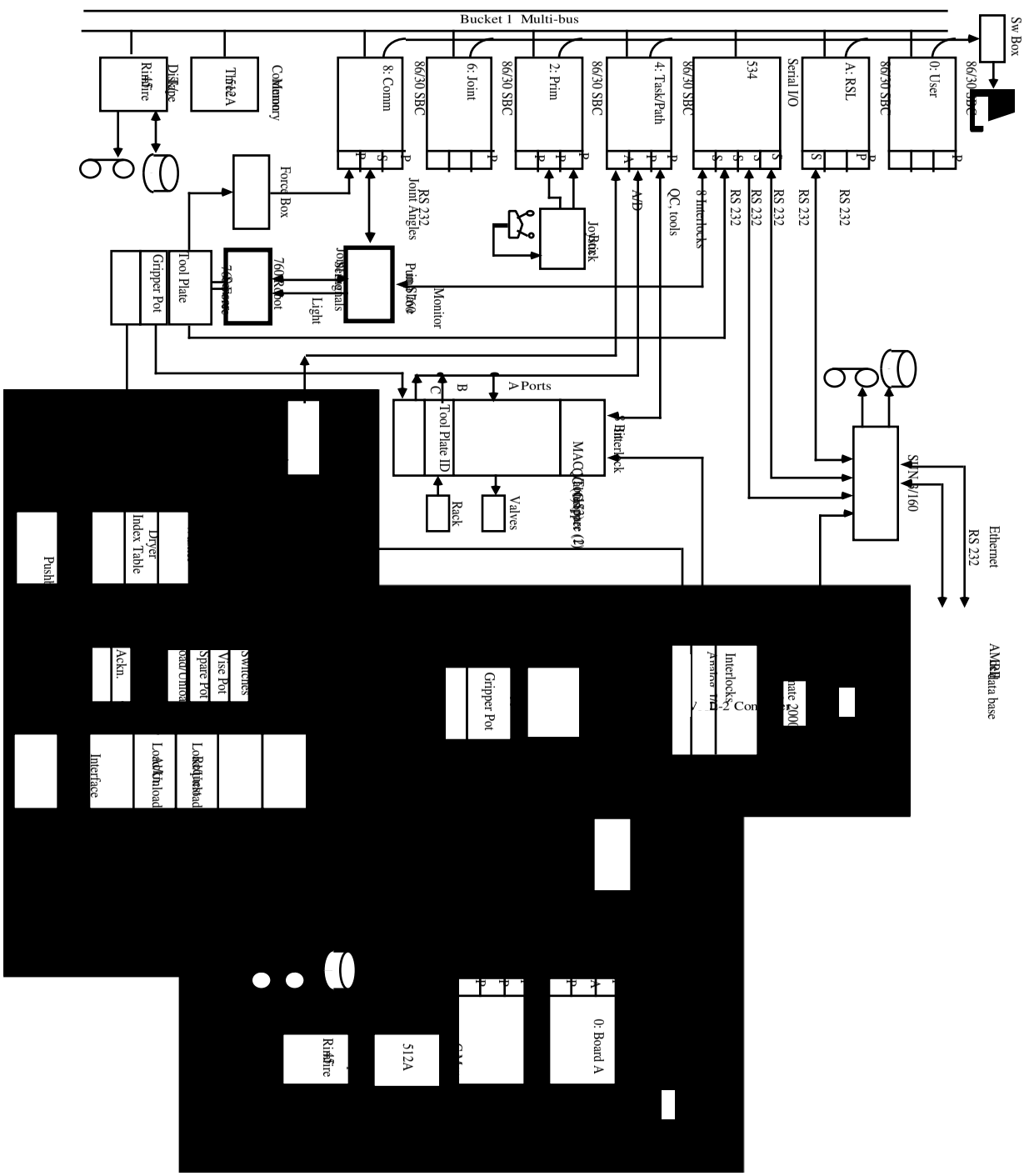
5) At Base of Unimate 2000 Robot, push red-handled circuit breaker to "OFF" position to disconnect power to Unimate 2000 Robot Controller.

6) At Acme Pushbutton Panel, push the red "STOP" button in to cut power to the equipment. At the Acme Power Panel, throw the power lever down to the "OFF" position.

APPENDIX A: WORKSTATION FLOOR PLAN



APPENDIX B: CONTROL SCHEMATIC



APPENDIX C: UNIMATE 2000 INTERLOCKS

	Command		Status
1.	vise req 0	1.	vise ack 0
2.	vise req 1	2.	vise ack 1
3.	acme req 0	3.	acme ack 0
4.	acme req 1	4.	acme ack 1
5.		5.	wash/dry/index error
6.		6.	
7.		7.	vise part error
8.		8.	gripper part error
9.		9.	
10.		10.	
11.		11.	L/U error
12.	index 0	12.	rot-vise error
13.	index 1	13.	force sensor health
14.	washer on	14.	force exceeded
15.	dryer on	15.	comp/man sw
16.		16.	Bucket2 health
17.		17.	index 0
18.		18.	index 1
19.	L/U req TS1	19.	L/U ackn TS1
20.	L/U req TS2	20.	L/U ackn TS2
21.	rot 180	21.	rot sw 180
22.	rot 90	22.	rot sw 90
23.	vise close	23.	vise pot closed
24.	force on	24.	gripper pot closed

APPENDIX D: LIST OF REFERENCES

[1] K. Murphy, R. Norcross, P. Tanguy, and F. Proctor, "Cleaning and Deburring Workstation Operations Manual", National Bureau of Standards Internal Report 88-3804, June 9, 1988.

[2] F. Proctor, "Technical Specifications of the Cleaning and Deburring Workstation", National Institute of Standards and Technology Internal Report (to be published).

[3] Unimation Inc., "Programming Manual: User's Guide to VAL-II Version 2.0 (398AG1)", December 1986

[4] Unimation Inc., "2070 Series Electrical Drawing Set (394K1)", November 1983

[5] R. Kilmer and S. Leake (editors), "The NBS Real-Time Control System / User Reference Manual" (to be published).

[6] Acme Manufacturing Company, "User's Reference Manual for Machine Job M-21309", August 1987

[7] L. Brodie, "Starting FORTH", Prentice-Hall, 1981.

[8] Lord Corporation, "Installation and Operations Manual: F/T Series Force/Torque Sensing System", March 1986

[9] Unimation Inc., "2070 Series Equipment and Programming Manual (398M1)", February 1984

Note: Schematics and mechanical drawings that have not been published may be available from the Robot Systems Division of NIST. Questions regarding unpublished technical information should be directed to:

Systems Integration Group
Robot Systems Division
National Institute of Standards and Technology
Gaithersburg, MD 20899
(301) 975-3422