N93-11982

A SMART END-EFFECTOR FOR ASSEMBLY OF SPACE TRUSS STRUCTURES

William R. Doggett; NASA Langley Research Center; MS 152D;Hampton Va 23665-5225
Marvin D. Rhodes; NASA Langley Research Center; MS 199; Hampton Va 23665-5225
Marion A. Wise; Lockheed Engineering and Sciences Company; Hampton Va 23665-5225
Maurice F. Armistead; Planning Research Corporation; Hampton Va 23665-5225

ABSTRACT

A unique facility, the Automated Structures Research Laboratory, is being used to investigate robotic assembly of truss structures. A special-purpose end-effector is used to assemble structural elements into an eight meter diameter structure. To expand the capabilities of the facility to include construction of structures with curved surfaces from straight structural elements of different lengths, a new end-effector has been designed and fabricated. This end-effector contains an integrated microprocessor to monitor actuator operations through sensor feedback. This paper provides an overview of the automated assembly tasks required by this end-effector and a description of the new end-effector's hardware and control software.

INTRODUCTION

NASA Langley Research Center has initiated a research program to develop methodologies for automated in-space assembly of large truss structures. A laboratory test facility, called the Automated Structures Assembly Laboratory, has been recently developed in which a regular tetrahedral truss structure, that includes 102 truss members, has been assembled using a specialized end-effector mounted on an industrial robot arm. A description of the truss is included in reference 1 and a description of the facility is included in reference 2. The end-effector currently used in the test facility was designed for, and is restricted to, operations on fixed length struts. Fixed length struts restrict operations to the assembly of structures composed of regular tetrahedral subelements similar to the current test structure. To address the assembly of other structures, such as support trusses for antennas that have a parabolic shape and require members of many different lengths, and to expand the capability of the system, to perform tasks such as the installation of payloads, a new end-effector is required. Also, for the current assembly experiment, the computational requirements have become very large and a distributed computational capability that incorporates dedicated microprocessors to command actuators and verify successful operations through the interrogations of sensors is needed. Therefore, an end-effector which installs one end of a strut at a time and uses a dedicated on-board microprocessor for sensor interrogation and actuator control, was designed and fabricated. This new end-effector will permit assembly of a variety of structures, installation of payloads with a standardized connection, and will provide the first opportunity for concurrent operations within the test facility. The time required to assemble a structure with the new end-effector is expected to increase over that required by the current end-effector, because the new end-effector operates on one end of a strut at a time and is required to capture most struts in a cantilevered configuration.

The purpose of this paper is to describe the new end-effector and the operations involved in truss assembly, describe the relationship between the end-effector and the overall hierarchical control scheme used in the assembly facility, and detail the operation of the end-effector's control software; highlighting the advantages that can be achieved through embedded microprocessor control. Emphasis will be placed on the unique features of the software design. Planned tests and further work will also be discussed.

STRUCTURAL ASSEMBLY TEST FACILITY

The current automated assembly test facility at the Langley Research Center is shown in the photograph of figure 1. Figure 1a is a schematic of the assembly system with the subcomponents labeled and figure 1b is a photograph of the actual assembly laboratory. The facility is a ground-based research tool to permit the development and evaluation of assembly hardware concepts, construction techniques, software; and operator interface systems that are likely to be required for on-orbit assembly operations. The robot arm is an electrically controlled, six degree-of-freedom industrial model that was commercially available when the program was initiated. The arm was selected for the laboratory test operations primarily because of its payload capacity, reach envelope, and positioning repeatability. No modifications to the robot other than those that are commercially available from the manufacturer have been made. The robot is mounted on an x-y Cartesian motion base that provides translational travel to position the base of the robot anywhere in the support track area to within 0.002 of an inch. The truss is mounted and assembled on a rotating motion base at the end of the translational base. Both of the motion bases are designed to minimize positional errors that may be induced by static deformations from the mass of the robot, unbalanced asymmetric truss assembly, and forces exerted by the robot during assembly.

The truss selected for assembly test operations is a regular tetrahedral truss composed of 102 strut members each approximately two meters long. This size and configuration were chosen because they are representative of the support structures that are required for a number of planned and/or proposed missions. The truss also has a number of geometric characteristics that make it desirable for an automated assembly test bed. These characteristics are described in reference 3. The truss members are connected by specially designed connector joints located near the nodes. Each node must be capable of connecting nine members, six that are in the plane of the top or bottom face and three core members that attach the top and bottom faces. The joints are located as close to the nodes as physically practical to minimize the packaging volume of the members for launch. The confined region near the node, however, somewhat complicates assembly operations since the end-effector must have a small size to be capable of installing a member. The truss joint connector is shown in figure 2. The joint is composed of two parts; a connector section that is bonded to the strut tube, and a receptacle section that is mechanically attached to the node. The joint has a ramped entry way to aid in inserting the connector into the receptacle. After insertion, the locking nut is turned to draw the connector plunger into the receptacle pocket and to preload the joint to eliminate free-play in the truss, thus providing a structurally predictable assembly unit.

The end-effector shown in figure 1 is the first generation model developed especially to assemble the truss shown. The initial tests were developed around the use of this specialized tool to establish a test database and provide experience for the development of the operator interface while the facility hardware was coordinated and tested.

The facility is directed and controlled by several digital computers that are currently serially connected through RS 232 communication lines to transfer commands and status information as shown in figure 3. For the current system all end-effector operations are controlled by the robot computer. This configuration was adopted because the robot computer was capable of directly monitoring end-effector sensors and executing commands through integrated analog-to-digitial and digital-to-analog converters. The status of the assembled structure, location of all truss members, position of the robot arm and motion bases, and status of the end-effector are all stored in appropriate databases. At the highest level, the operator selects the desired function to be performed from a menu of permissible commands. The executive program checks the current status of all components and the status of the assembled structure and determines if the requested command is permissible. All assembly operations are performed under the supervision of the computer executive, which operates as an expert system. During the assembly process, a display adjacent to one of the menus advises the operator of commands initiated and their completion status.

Shown in figure 4 is the assembly software structure which is divided into five hierarchical levels: planning, truss element, device, component and verification. The figure indicates typical operator commands at each level, which are decomposed by the automated system into a series of commands for the next lower level. The highest, or planning level, which is not currently implemented, would be driven by an automated task sequence planning tool. A command to "build" a structure or substructure would be decomposed by the planner into a sequence of commands for the second level. To fetch and install a strut requires action by 3 separate devices; the motion bases, the robot, and the end-effector. Thus, the truss element decomposition is sequentially directed toward separate targets. Each of the device commands, such as the end-effector install command (INSTALL) decomposes into a sequence of individual actuator commands (CLOSE, LOCK, EXTEND,...). If a failure is detected, automated error recovery is activated. If the automated error recovery is not successful, an error recovery menu is displayed to the operator. On the menu is a list of methods to overcome the failure. The order of the list and the definition of the methods are based on experience. The system will not proceed until the problem is resolved. If it cannot be corrected, information is passed back up through the system hierarchy, causing all commanded actions to roll back to the state that existed prior to initiating the command. More information regarding fault corrections and overrides, especially with regard to the end-effector, will be discussed in a subsequent section. Further details on the facility components and the software system can be found in reference 3.

DESCRIPTION OF ASSEMBLY HARDWARE

The second generation end-effector discussed throughout this paper is a follow-on to the first generation end-effector used to assemble the planar structure shown in figure 1. This end-effector is shown in the center of figure 5 surrounded by enlargements of the end-effector components. The second generation end-effector is much smaller than the first generation model, with a length of about 0.52 meters (20.5 inches). It is designed to insert one end of a strut at a time. The receptacle fingers, nut driver, insertion platform, and strut holder are adopted directly from the first generation end-effector. The second strut holder (figure 5b) and alignment fingers (figure 5c) were added to increase the moment capacity and capture zone of the end-effector. The alignment platform permits the alignment fingers to be extended away from the strut holders when capturing cantilevered struts.

The end-effector contains four grippers, two platforms, a nut driver, and seventeen sensors (not shown in the figure). The insertion platform, shown in figure 5e, is used to insert the strut connector into the node receptacle. The alignment platform is used to extend the alignment fingers to prevent interference with other end-effector components when capturing a cantilevered strut, as mentioned above. Referring to figures 5e and 5f both platforms will extend and retract. These operations are accomplished by air cylinders with full extension and retraction verified by linear potentiometers. Figures 5a and 5c depict the operation of the receptacle and alignment fingers. The receptacle fingers align the end-effector relative to the structure by closing on the vee grove of the node receptacle, the alignment fingers are used to capture a cantilevered strut by closing on the strut tube. Both the receptacle fingers and the alignment fingers are actuated by sliders driven by air cylinders. Position of the cylinder piston is monitored via the manufacturer's reed switches mounted on cylinder exterior. The reed switches are activated by a permanent magnet internal to the pneumatic cylinders. The receptacle and alignment fingers also contain infrared cross-fire sensors to verify that a strut is in the capture zone. The cross-fire sensors operate using a pair of units that project light from an emitter unit to a receiver unit. A receptacle or strut is detected when it blocks the light. Figure 5b depicts operation of the strut holders. The strut holders are driven by DC motors through a screw jack mechanism. The strut holders align the end-effector relative to the strut by closing on the strut's alignment adapter. Motor power is applied to the strut latch motors until a sensor is activated or the time limit is exceeded. Three inductive sensors are used to monitor each strut holder's state: one sensor is used to indicate that the holder is FORCED CLOSED, and two to sense the state of the motor-driven latching mechanism, either DRIVEN OPEN or DRIVEN CLOSED. The strut holders are spring loaded to the open position. FORCED CLOSED occurs when the robot is commanded to move until the strut holders are forced closed by contact with the alignment adapter located on the strut, then the strut holders are DRIVEN CLOSED securing the strut to the end-effector.

The nut driver, figure 5d, is also driven by a DC motor. The nut driver is used to actuate the joint connector. The nut driver is driven using a closed loop control system with current and position feedback. Current feedback is derived from the voltage drop across a resistor in series with the motor winding. Position feedback is achieved by a low resolution encoder on the output shaft providing eight pulses per revolution. Since the flats of the socket and the nut are randomly oriented during acquisition of the strut, the socket will not necessarily slip onto the nut, but instead may push the strut away from the holders, preventing them from grasping the strut. The socket is therefore spring loaded toward the strut to permit grasping of the strut even though the nut driver may not be seated. Socket seating is monitored by an inductive sensor.

STRUT INSTALLATION

The second generation end-effector must be able to install struts into nodes supported by one or more previously installed struts. As an example of the operations required for strut installation, a possible double cantilever installation scenario will be developed. In this case the end-effector has installed one end of a strut and now must connect the other cantilevered end to an existing node at the end of a second cantilevered strut. A cross-fire sensor verifies that a strut is within the capture zone of the alignment fingers, figure 6a, the alignment fingers are closed on the cantilevered strut's tube restricting the translation of the strut in two directions. With the strut captured and the proper alignment achieved, figure 6b, the insertion platform is extended and strut holders latched securing the strut to the end-effector, figure 6c. The left strut holder closes around the hexagonal alignment adapter that is bonded to the strut tube. These adapters have a central vee groove that mates with a protrusion in the left strut holder to provide passive axial alignment, while the hexagonal cross section of the adapter keeps the strut circumferentially aligned. The right strut holder closes on the strut tube to increase the moment capacity of the end-effector's grasp. At this time the alignment fingers are opened, figure 6d, followed by the retraction of both platforms. With the cantilevered strut secure, the next step is the capture of the node at the end of the second cantilever strut. After verifying the receptacle is within the capture zone of the receptacle fingers using the cross-fire sensor, the fingers are closed on the vee groove machined into the connector receptacle, passively correcting small misalignments. The end-effector fingers are designed to successfully capture the receptacle within a cylindrical envelope of 5 cm diameter by 1.5 cm long. With the receptacle captured, figure 6e, the insertion platform is extended inserting the joint connector into the receptacle. The nut driver socket is seated; if required, by incrementally turning the socket to align it with the nut, then several full turns of the nut securely lock the joint. While the joint is being locked motor current, encoder counts, and socket seating are monitored. Normal termination occurs when current draw exceeds a predefined threshold and a minimum number of encoder counts have been received, while the socket has remained seated. Reaching the current threshold indicates the locking mechanism has applied adequate closure torque to the joint connector, while the encoder count indicates that the joint connector is fully closed. This operation secures the strut in the structure. Next, the strut holders are unlatched releasing the strut, the insertion platform is retracted, and the receptacle fingers are opened, releasing the structure and completing the connection. All operations are verified using sensors as discussed earlier. All other installation cases are based on variations of these operations.

The double cantilever installation is a potentially difficult task and several operational scenarios are possible. For example, the strut with the node may be captured first. The purpose of the above illustration is to demonstrate how the operation of the end-effector components must be coordinated to perform an installation sequence and how sensors are required to verify the operation of the end-effector components. It is important to note that all mechanical operations must be totally reversible to permit the end-effector to recover from an error or to permit the truss to be disassembled for repair.

END-EFFECTOR CONTROL SOFTWARE

The driving force for development of an on-board embedded microprocessor has been to reduce the number of signal lines from the end-effector to accommodate a quick change mechanism. The end-effector is essentially a tool for the robot and, since the robot is expected to perform many operations, it occasionally requires different tools. This led to the purchase of a quick change mechanism to allow the robot to engage or disengage end-effectors. But, this piece of hardware places a constraint on the number of signal lines between the end-effector and the robot. Thus, a method for integrating the sensors with the end-effector operations was required to accommodate the increasing number of sensors on the end-effector. This need lead to the development of the microprocessor based control scheme discussed in this paper.

An assessment of the requirements for the microprocessor system was conducted and the following microprocessor capabilities were identified: (1) a minimum number of signal lines passing through the quick change mechanism, (2) small physical dimensions, (3) standard communication support, (4) six input lines for analog to digital conversion, (5) two output lines for digital to analog conversion, (6) twenty-four general purpose I/O pins, and (7) three lines providing external interrupt capability. In addition, high level language and development tools were required to support the implementation of the following software capabilities: (1) stand alone checkout, (2) on line checkout, (3) communication interrupts, (4) operator override, and (5) eventual porting to different computer platforms.

These requirements led to the selection of a commercially available single board computer built around an eight bit microprocessor operating at a clock frequency of 12MHz. The board supports up to eight general purpose eight bit ports including two serial communication lines, external interrupts, data bus, and address bus. The board also supports eight analog to digital lines with programmable reference voltages and up to 64 kbytes of RAM and 64 kbytes of ROM.

To minimize the number of signal lines leaving the end-effector and provide necessary signal preparation a custom electronics board was fabricated to interface between the end-effector actuators/sensors and the microprocessor board. This board provides: (1) electronics for digital to analog conversion for proportional nut driver motor control, (2) electronics for analog to digital conversion to monitor nut driver motor current, (3) the drive electronics for the strut latch motors and solenoid valves, (4) electronics to convert plus and minus 24 volt power to the additional voltage levels required for the microprocessor and the drive electronics circuits, and (5) isolation of all sensor and actuator signals from the microprocessor ports by Schmitt triggers or voltage followers. By locating the support electronics on the end-effector, the number of signal lines required to operate the end-effector has been reduced from 60 to 5, providing enough free lines in the quick change mechanism for two color camera cables used by the operator to monitor end-effector operations.

The software was implemented in C using a PC based development system. The system was selected because it was ANSI C compatible, included a source level debugger, and included language extensions providing access to all processor dependent features. C was selected because of its support for bit level operations and to support transferring developed code to other computers.

The relationship of the end-effector to the overall assembly configuration is shown in Figure 7. Note that the second generation end-effector will communicate with the executive computer directly instead of through the robot as was depicted in figure 3. The end-effector control software is responsible for three levels of operation that were depicted earlier (figure 4); device, component, and verification. The software implementation focused on support for event driven operations. Event driven refers to program flow dictated by responses to events rather than moving serially through a state system. Event driven applications have many advantages. In general they provide improved response time, support asynchronous events, and provide interrupt capability. The ability to support asynchronous events and provide interrupt capability were essential for this application.

The relationship between end-effector commands and software states is shown in figure 8. Figure 8a is the state transition diagram for the end-effector software, while figure 8b is the corresponding transition matrix. In figure 8a arrows represent possible transitions, dashed arrows represent interrupts, and bold arrows represent normal operation.

Commands associated with a given transition overlay the arrow representing the transition. The shaded box on the left encloses the states making up the control loop. All commands that affect end-effector actuators occur within this box. The shaded box on the right side of the figure encloses the three states within the command parser. The software transitions from the box on the left to the one on the right via an interrupt activated by the reception of a command. When power is applied to the microprocessor's board, or the board's reset button is depressed, the software initializes (figure 8a) the actuators, based on sensor information, and enters the control loop state WAIT FOR COMMAND. Under normal operation, the software is in the control loop in state WAIT FOR COMMAND when an executable command is received across the serial line activating the serial interrupt handler (command parser). The command parser reads the command, validates the command based on the current state, and initiates the transition from WAIT FOR COMMAND to EXECUTE COMMAND. After execution of the command has completed, the software waits for another command. The software design was governed by the possibility of being interrupted at anytime. For this reason and because the command parser initiates the transition from state WAIT FOR COMMAND to state EXECUTE COMMAND, the command parser and the commands it accepts will be discussed first.

The command interface to the microprocessor was designed to provide a human readable ASCII format. This greatly simplifies diagnostic and monitoring operations by creating data that can be immediately read and interpreted by the operator. This interface has the added advantage of supporting dumb terminals for off-line checkout and verification. For this verification to have relevance, it should mimic the conditions of the actual system. Thus, the commands issued from the terminal must be the same as those issued to the end-effector at run time. By carefully defining the syntax of the end-effector commands, the overhead is minimized while maintaining a human readable format. Under normal operation the maximum command contains seven ASCII characters.

The most critical requirement for asynchronous event support was to allow PAUSE and REVERSE commands at any point. PAUSE implies suspending the current operation indefinitely. REVERSE implies removing the progress of the current operation until the state of the system is the same as it was before the command was received. As shown by the dashed lines in figure 8a, command reception activates the command parser. While the software is in the state EXECUTE COMMAND, reception of a PAUSE command will cause the software to enter the PAUSED state suspending execution indefinitely. Either a CONTINUE or a REVERSE command is required to resume execution.

The correspondence between the commands and software state transitions is shown in figure 8b. The states that can be interrupted are located on the left of the figure, the acceptable commands in the center, and the resulting state on the right. Commands are interpreted based on the current software state, thus there are three cases of command evaluation within the command parser, one for each software state that may be interrupted by command reception. Interruption occurs when input is detected on the serial line, because the command parser (serial interrupt handler) is activated to process it, thus, allowing commands to be received at anytime. Note that the command parser contains all decision points related to software flow.

Supervisory commands can be entered from any state and do not cause a state transition. Supervisory commands provide a method for direct operator interaction with the end-effector. For example, a single step mode may be turned on. This causes the end-effector control software to execute component commands individually. After completion of each component command, the operator is queried to continue. Supervisory commands also support diagnostic operations. They give the operator a way of requesting end-effector status information to monitor individual sensors. Consider the platforms discussed earlier; if they indicate intermittent failures when a platform actually worked, then the platform potentiometer may be monitored to determine if the thresholds representing fully extended or retracted need to be adjusted. After completion of the supervisory action, control returns to the state that existed prior to the reception of the supervisory command.

While in the software state WAIT FOR COMMAND, executable commands are accepted. Executable commands initiate end-effector action, thus causing transition to the state EXECUTE COMMAND. For example, LOCK NUT is an executable command that causes the nut motor to turn. CONTINUE and REVERSE are also accepted in the state WAIT FOR COMMAND. Reception of these commands implies the previous command encountered an error, because an unresolved error terminates command execution. CONTINUE implies the error has been overcome and execution is to resume, REVERSE implies the error was not overcome and the system must be rolled back to the state before the command was received.

From the state EXECUTE COMMAND, reception of REVERSE implies the operator has foreseen a potential problem and wishes to retract the last EXECUTABLE command. If PAUSE is issued by the operator, while in the end-effector software state EXECUTE COMMAND, the end-effector will suspend its current operation prior to entering the PAUSED state. To suspend the current end-effector operation, the control software uses a queue of pause functions. These are functions queued by the component or composite functions to be activated anytime a PAUSE is received. For example, if the PAUSE command is received while locking the nut, the nut driver motor must be stopped. This is accomplished using a function queued by the nut component that stops the motor and saves the encoder counts. After activating each function in the pause queue, the control software enters the PAUSED state. The end-effector may remain paused indefinitely. As shown at the bottom of figure 8b, execution of the interrupted end-effector operation, i.e. return to the state EXECUTE COMMAND from the state PAUSED, will occur only after reception of a CONTINUE or REVERSE command. Queues similar to the pause queue are maintained for CONTINUE and REVERSE. If CONTINUE is received, continuing from the PAUSED state entered above, a function in the CONTINUE queue is activated to restore the encoder counts and restart the nut motor, thus returning to the state before the PAUSE command was received. If an invalid command is entered at any time an error is returned.

The control loop, shown on the left of figure 8a is shown in detail in figure 9, with the corresponding portions of the software flow diagram enclosed in circles. The block ACTIVATE FUNCTION is further detailed in the lower right of the figure. After power is applied to the microprocessor board or a reset occurs, the end-effector software outputs a message, signaling successful initialization, and waits for a command to execute. At this time the software is in an infinite loop, constantly checking a flag that, when TRUE, indicates a command is available to execute. This flag can only be set to TRUE through an interrupt handler. As mentioned above, the command parser is an interrupt handler. By setting the flag to TRUE after a valid command has been received, the command parser initiates the transition from the state WAIT FOR COMMAND to the state EXECUTE COMMAND.

Commands are of two types, component and composite. Both are activated and processed in the same fashion within the control loop as shown in figure 9. Component commands interact with the hardware directly and hence, are specific to a given end-effector. All component commands contain a timed loop during which a specific sensor is polled to verify success of the operation. Three basic capabilities are provided for each end-effector component by component commands. These capabilities are: (1) specifying the desired configuration of the component, for example EXTEND INSERTION; (2) exercising the component, for example CYCLE INSERTION, which attempts to extend and then retract the insertion platform; and (3) analyzing the component function, for example SURVEY INSERTION, which cycles (for a predefined number of times) the insertion platform and displays a series of formatted strings containing the commanded state, the thresholds representing fully extended and retracted, and the current value of the linear potentiometer.

An example component function block diagram is shown in figure 10. This component is responsible for receptacle finger operations. For each command available for the component, a section of code is required to define the command operation. In this example the executive command CLOSE RECEPTACLE was received. Upon entering the section of code for CLOSE, a sensor check is made for receptacle presence via infrared cross-fire sensors. An error is generated if the receptacle is not present. If a receptacle is present then the actuator bit corresponding to the receptacle fingers' actuator is set. The software then enters a timed loop during which the reed switch, which indicates closed, is constantly polled. If the receptacle fingers close before the allotted time has expired the operation is successful, and a message indicating success is returned by the control loop. Time limits used in the component commands were established empirically based on experience with the first generation end-effector. If the receptacle does not close within the allotted time an error code is returned.

In the command hierarchy, composite commands are above component commands. Composite commands are a set of end-effector specific component commands executed in a predefined order. Composite commands allow for a greater level of abstraction, thus, they provide a method for increasing the embedded intelligence at the end-effector level. Composite commands were established to provide a device independent interface to the end-effector operations. Use of composite commands has significantly reduced the code requirements in the executive program by standardizing the interface to all end-effectors. Also, by supporting the composite commands at the end-effector level, most of the communication overhead has been eliminated. The block diagram for an example composite command is shown in figure 11. This is the sequence for a subset of the double cantilever INSTALL function. In the explanation that follows, it is assumed that the end-effector already has a strut latched by the strut holders and the insertion platform is retracted. Because any command performed by the end-effector must be reversible, each composite function has a corresponding reverse function, in this case INSTALL_REVERSE, to return the end-effector to the state prior to reception of the composite command. Both functions are depicted in the figure; INSTALL on the left, INSTALL_REVERSE on the right. An INSTALL operation with no errors will enter at the top, the continue check will be FALSE, and the operations from SET_UP to COMPLETE will occur sequentially. In SET_UP the receptacle is verified to be present, then the receptacle fingers are closed in CLOSING_RECEPTACLE. Next a message is sent to the executive program requesting a force/torque balance. Force/torque balancing will be used to adjust the robot arm to correct for misalignments between the end-effector and receptacle slot that prevent full extension of the insertion platform. The robot is adjusted until all reaction forces between the robot and structure have been reduced to near zero. At this point the end-effector will be in the configuration shown in figure 2e. As discussed previously, the insertion platform is then extended and the nut locked to secure the strut. Finally the strut is unlatched, the insertion platform retracted, and the receptacle fingers opened, releasing the structure and completing the installation sequence. A sensor is polled after each operation to verify successful execution. Because no resources have been allocated, no operations occur in COMPLETE. The software then returns to the control loop where a message indicating success is returned to the executive program.

Software supporting error handling has dominated the implementation efforts for the composite commands. In general, approximately 70% of the software is required to support error conditions. If an error occurs in one of the component operations; the function is terminated, returning to the main control loop where a descriptive error message is returned to the executive program. After receiving an error message the executive program presents an error menu to the operator through which the operator can: (1) fix the error and continue, (2) reverse the current command, or (3) override the error and continue. It is important to note that, after an error is encountered, the function is terminated and the end-effector software is in the control loop state WAIT FOR COMMAND. This provides two advantages. First, since there is only one command parser, each function does not have to support communications during error handling. Second, there is no restriction on the commands that can be issued when trying to resolve an error. This is critical, because it is often necessary for the operator to activate other end-effector components or command other devices when resolving an error. For example, when closing the receptacle, the end-effector's location may need to be modified by commanding the robot to move. Thus, the receptacle is opened, using a component command, the robot moved, and the installation sequence resumed by a CONTINUE command.

If the operator is able to eliminate the error by adjusting some assembly component, then a CONTINUE command is issued to the end-effector. The control software sets the continue bit too TRUE, causing the CONTINUE? check at the beginning of INSTALL to be TRUE, thus returning to the state within the INSTALL function where the error occurred and continuing execution from that state.

If the operator is unable to overcome the error, the REVERSE command is sent to the end-effector. The end-effector software sets the flag indicating CONTINUE, but also sets a flag causing the components to abort. When the last state is entered, it is immediately aborted, returning a unique error condition to the INSTALL function. The INSTALL function clears the error before calling the INSTALL_REVERSE function. INSTALL_REVERSE jumps to the current state and begins reversing the operations completed by the INSTALL function. If another REVERSE command is received, then the same error is cleared by the INSTALL_REVERSE function and control is returned to the INSTALL function. This cycle can repeat indefinitely. Jumps to the current state are depicted by large arrows under the line TO CURRENT STATE at the bottom center of figure 11. Note that some of the states in INSTALL_REVERSE have no actions associated with them, for example BALANCING_FTS, which is only required after the end-effector has grasped the structure. These states are simply place holders so that when reversing there is an appropriate state to jump to. An example of a reverse sequence follows. While in the INSTALL state CLOSING_RECEPTACLE, the receptacle fingers fail to close, the timing loop initiates an error causing the INSTALL function to terminate and return to the control loop, where an error message is sent to the executive program. The end-effector software then enters the state WAIT FOR COMMAND. Because the operator is unable to overcome the error, the REVERSE command is sent to the end-effector. The end-effector control software interprets the REVERSE command, sets the abort flag to TRUE, sets the CONTINUE flag to TRUE, and activates the INSTALL function. Upon entering the INSTALL function the CONTINUE check is TRUE, causing execution to return to the state

CLOSING_RECEPTACLE. When the component command responsible for receptacle operations is executed, it immediately aborts, because the abort flag is TRUE, returning a unique error code to the INSTALL function. The INSTALL function clears the error before calling the INSTALL_REVERSE function, thus reversing the sequence. While in the INSTALL_REVERSE function, reception of another REVERSE command will result in the same error code being cleared by the INSTALL_REVERSE function causing execution to return to the INSTALL function. Cycling between INSTALL and INSTALL_REVERSE can repeat indefinitely. The REVERSE command may also be received while the software is in state EXECUTE COMMAND. Reception of the REVERSE command from this state results in the abort flag being set to TRUE causing the current component operation to abort. The unique error condition is cleared and the REVERSE function is executed as described above.

Finally, it is possible to override an error and continue the execution of the command. This capability is provided by a function called UPDATE. UPDATE can be thought of as a background process, because it is never called directly. UPDATE operates as an interrupt handler cycling at 15 Hertz using an autoreload timer available on the microprocessor. UPDATE provides four basic functions: (1) a method of overriding sensor information, (2) support for asynchronous notification of hardware changes, (3) a method for monitoring sensor health, and (4) isolation of the software from the computer hardware to ease the transfer of the software to other computers. In addition UPDATE is the function responsible for interpreting the sensor data and updating the database, as well as interpreting the command settings and updating the database and end-effector actuator settings.

The capability to override sensor information, requires a database independent of the end-effector hardware. When power is applied to the microprocessor, or the microprocessor boards reset button is depressed, the software initializes the database based on the current end-effector configuration. Thus, the information in the database represents the current state of the end-effector hardware and software control flags. However, because this database is independent of the end-effector hardware, the information from the senors may be ignored by simply not updating the database. The M_VAX_CONTROL mask is used to prevent updates of the database and actuator settings. Bits set in this mask are not modified, thus indicating the operator has taken responsibility for the setting of one or more sensors. NULLIFY is the supervisory command used to initiate sensor override. The end-effector control program uses the last error status to set the database variable so that no error condition will be generated, and sets bits in M_VAX_CONTROL to prevent UPDATE from changing the value in the database. For example, if the receptacle fingers did not close, then this error will be overridden as follows. First the sensor bits in the M_VAX_CONTROL mask corresponding to the sensor which detect finger closure and the sensor that detects fingers opening are both set to prevent UPDATE from modifying these sensor bits in the database. Then the bits in the database are set directly to indicate the receptacle fingers are closed. The open sensor is set to FALSE and the close sensor is set to TRUE. Note that the order is critical to prevent conflicts which occur if both bits are TRUE simultaneously. At this point the database is configured so that the receptacle fingers appear closed to the software. The next command to close the receptacle fingers will succeed, thus successfully overriding the FAILED TO CLOSE error.

Support for asynchronous notification of hardware changes is provided by a third mask, the M_WATCH mask. If a watched value changes, a software interrupt is activated, thus notifying the control software of the change. For example, for correct operation of the end-effector, the supply voltages must remain nominally constant. Monitoring supply voltage could be implemented using the M_WATCH mask. If the voltage changes, the watch function is called to compare the voltage level to predefined thresholds. If the voltage is out of range the end-effector could be switched to battery backup and the executive program notified so that corrective actions could be initiated. This capability has not been implemented to date.

UPDATE provides the capability for monitoring sensor health. Currently only conflicts are reported. Conflicts occur between sensors that are configured so that both cannot be true simultaneously. For example, the receptacle fingers are designed so that they cannot simultaneously activate the two sensors that indicate CLOSE and OPEN. If this occurs, the hardware has failed, either mechanically or electrically, and this failure is reported to the host as a sensor conflict. This capability can be expanded to provide on-line sensor reconfiguration, which would require redundant sensing capability that is not available at this time. Note that; in general, two sensors that disagree provide only the information that the signal from one sensor is invalid because of some form of failure. There is no information about the state of the operation being sensed. If the operation of the system has been monitored, and the system has performed as expected, then the sensor that disagrees with the command signal is probably invalid, because it is unlikely that both a sensor and an end-effector operation will fail simultaneously. If three sensors are available; then the two that agree are considered valid, while the remaining sensor is considered invalid and flagged for maintenance.

UPDATE isolates the software executing on the microprocessor from the microprocessor specific features. It is one of two microprocessor specific functions. The other is used to initialize the microprocessor board after turning on power or after a reset. Thus only two functions need to be modified if it is desired to move to a different processor.

TESTS AND CURRENT OPERATIONAL STATUS.

Currently the end-effector hardware and software are undergoing integration tests in an off-line checkout mode. All support software, including the composite sequences, have been implemented. The end-effector will then undergo extensive testing on a six-axis servo table to refine the operational scenarios required to assemble the tetrahedral structure, including the double cantilevered capture and insertion operations described earlier. During this process, common errors will be logged along with their corrective actions. This will be used to add self-correcting logic to each of the composite commands. The outcome of these tests should finalize the component commands and the sequences used in the composite commands.

After development testing is complete on the six-axis servo table, the end-effector will be used to assemble the two ring structure shown previously in figure 1. Following successful assembly, operations will likely progress to the assembly of structures with curved surfaces, which are assembled using straight struts with different lengths.

Work has already begun applying these routines and techniques to other end-effectors used in the assembly facility. In support of these activities, generic code models are being developed for actuator/sensor groups. Not only will these techniques be applied to the end-effectors of the facility, but to other devices in the facility, such as the motion bases and pan tilt drive units to position video cameras. These distributed dedicated microprocessors provide an initial step toward the realtime distributed architecture needed for efficient control of automated assembly operations.

SUMMARY

NASA Langley Research Center has initiated a research program to develop methodologies for automated in-space assembly of large truss structures. A laboratory test facility, called the Automated Structures Assembly Laboratory, has been recently developed in which a regular tetrahedral truss structure, is comprised of 102 struts, has been assembled using a specialized end-effector mounted on an industrial robot arm. To expand applications that can be addressed by the facility, an end-effector which installs one end of a strut at a time and uses a dedicated on-board microprocessor for sensor interrogation and actuator control, was developed and fabricated. The new end-effector will permit assembly of a variety of structures, installation of standardized payloads, and will provide the first opportunity for concurrent operations within the test facility.

The new end-effector has been fabricated, and most of the control software written. The end-effector adopts many features from the first generation end-effector. The new end-effector contains four grippers, two platforms, and a nut driver. The end-effector is directly controlled by an embedded microprocessor. The software supports asynchronous interrupts to allow the operator to suspend or reverse end-effector operations at any time.

The end-effector control software development was dominated by the need to support error conditions and asynchronous events. Thus, efficient means of interrupting software execution and servicing asynchronous events are essential, both when interacting with human operators and when interacting with end-effector hardware. The end-effector software development effort has isolated and standardized the interface to the end-effectors used in the facility, resulting in a significant code reduction and generalization of the executive program.

The embedded microprocessor and support electronics board has reduced the number of signal lines required to operate the end-effector from 60 to 5, allowing the exchange of end-effectors via a quick change mechanism. Also, addition of an embedded microprocessor to the end-effector currently used in the facility is expected to significantly reduce assembly time by supporting concurrent operations and eliminating most of the time required for end-effector communications.

The second generation end-effector will greatly increase the flexibility of the Automated Structures Assembly Laboratory. This end-effector development effort represents a first step toward the realtime distributed computational architecture needed to provide efficient control for automated assembly systems.

REFERENCES

1) Wu, K. Chauncey; Adams, Richard R.; Rhodes, Marvin D.: "Analytical and Photogrammetric Characterization of a Planar Tetrahedral Truss", NASA Technical Memorandum 4231, December 1990.
2) Rhodes, Marvin D.; Will, Ralph W.; Wise, Marion A.: "A Telerobotic System for Automated Assembly of Large Space Structures", NASA Technical Memorandum, March 1989.
3) Will, Ralph W.; Rhodes, Marvin D.: "An Automated Assembly System for Large Space Structures", SPIE Symposium on Advances in Intelligent Systems, Cooperative Intelligent Robotics in Space, November 1990.

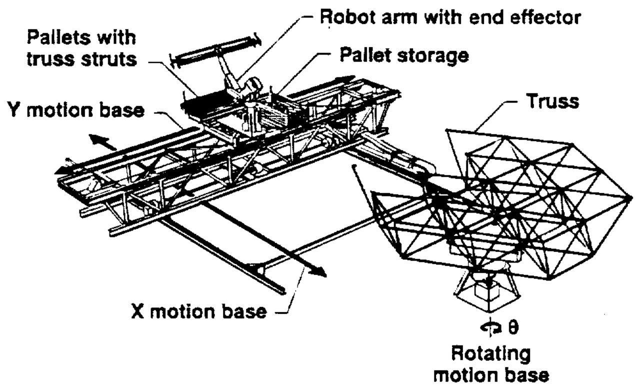

Figure 1a
Schematic of Automated Assembly Facility.

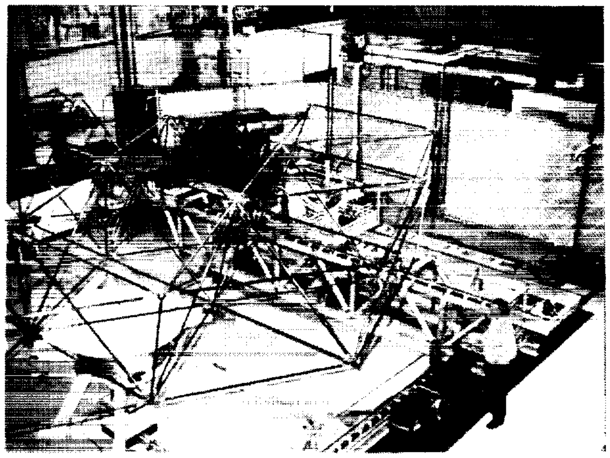

Figure 1b
Photograph of Assembly Facility.

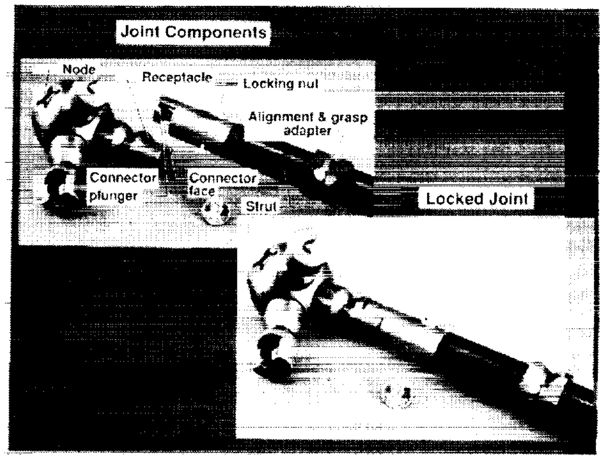

Figure 2
Photograph of Truss Node and Joint Connection Hardware.

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

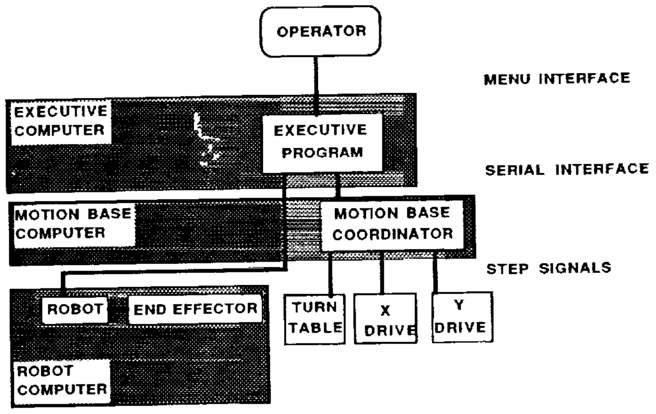

Figure 3
Current Assembly Facility Computer and Software Architecture.

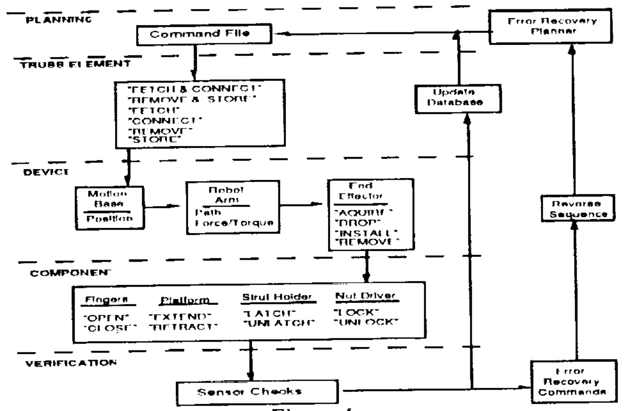

Figure 4
System Levels and Command Hierarchy

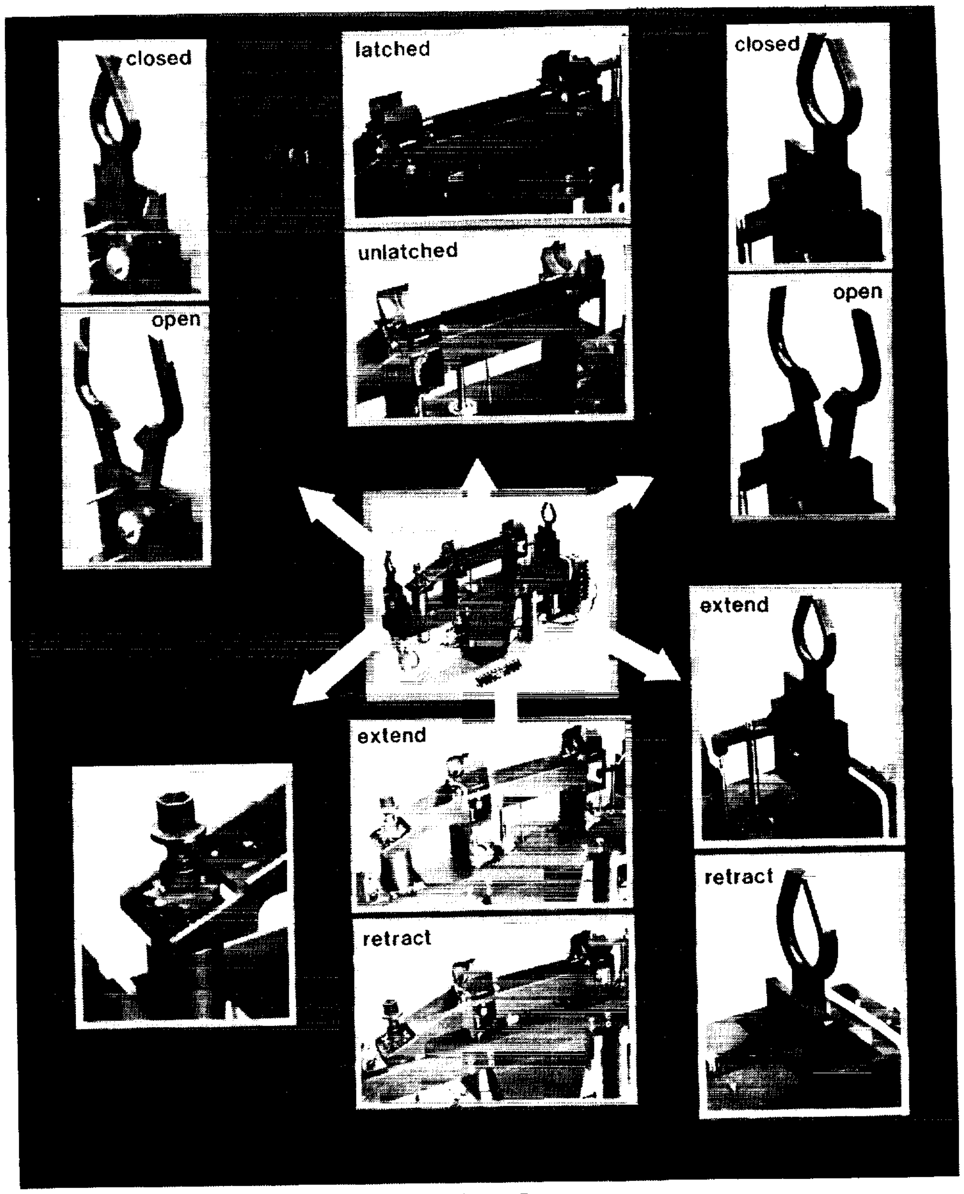

Figure 5
Second Generation End Effector.

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

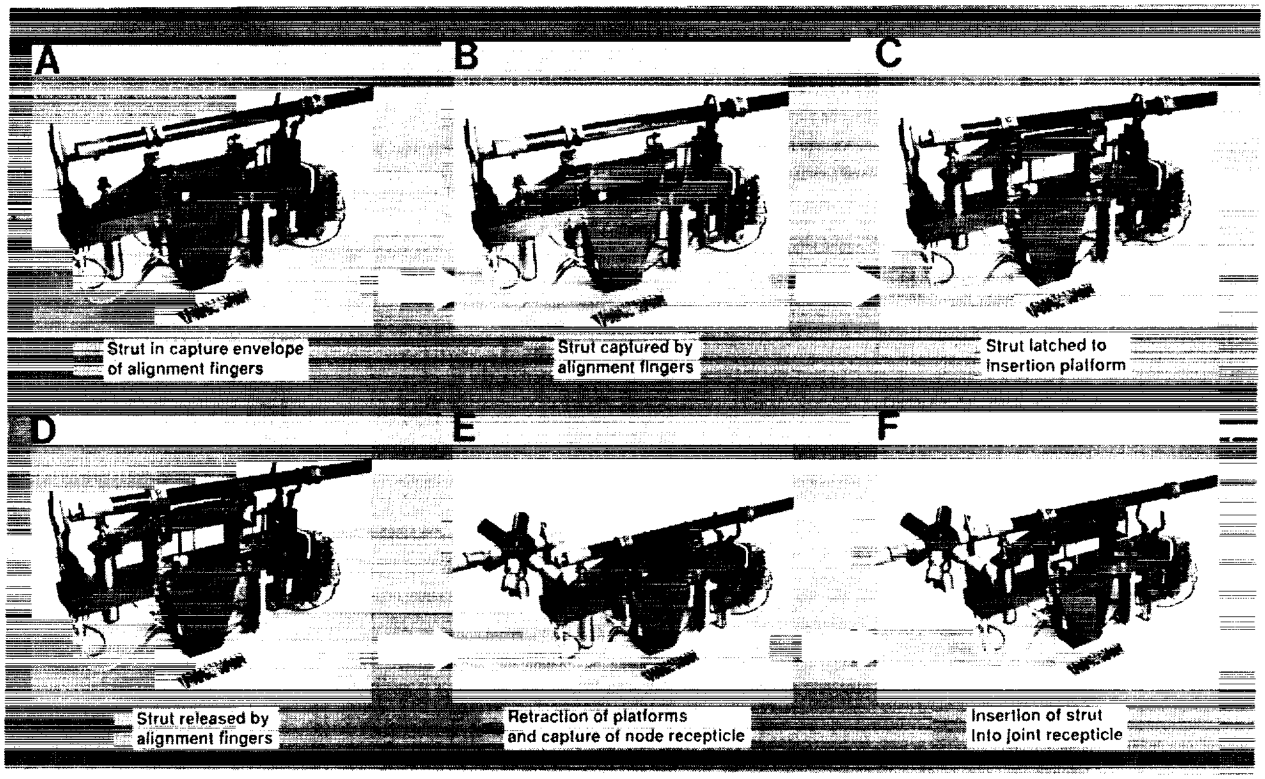
A
Strut in capture envelope of alignment fingers
B
Strut captured by alignment fingers
C
Strut latched to insertion platform
D
Strut released by alignment fingers
E
Retraction of platforms and capture of node recepticle
F
Insertion of strut into joint recepticle

Figure 6
Capture and Installation of a Strut.

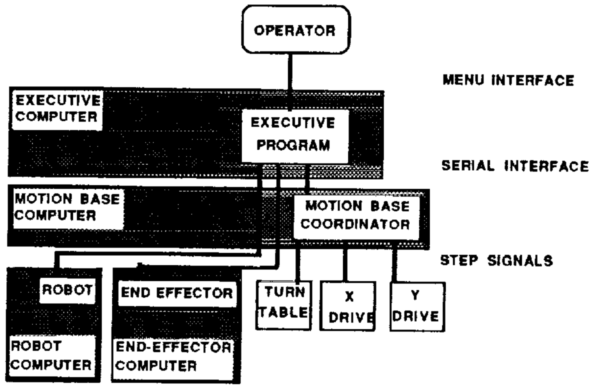

Figure 7
Current Assembly Facility Computer and Software Architecture.

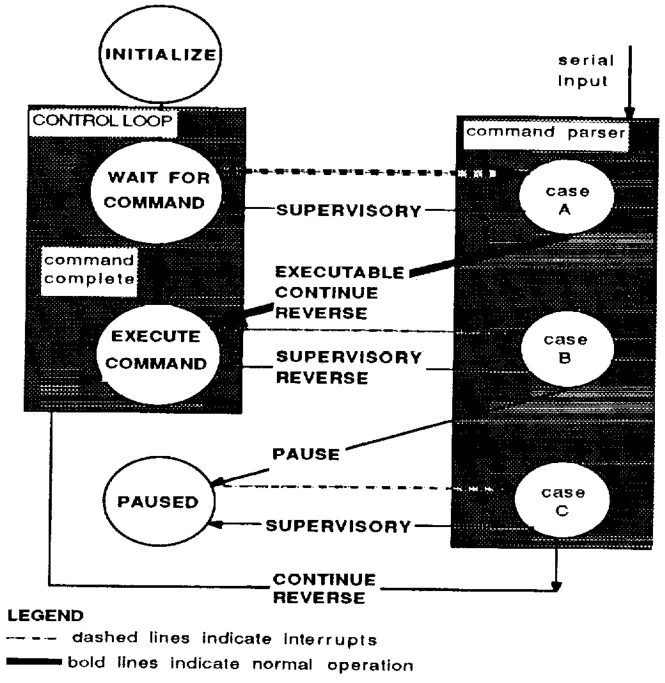

Figure 8a
Software State Transition Diagram.

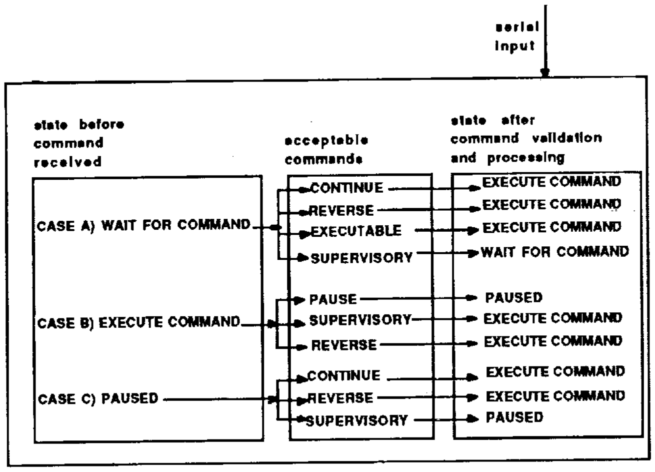

Figure 8b
Software State Transition Matrix.

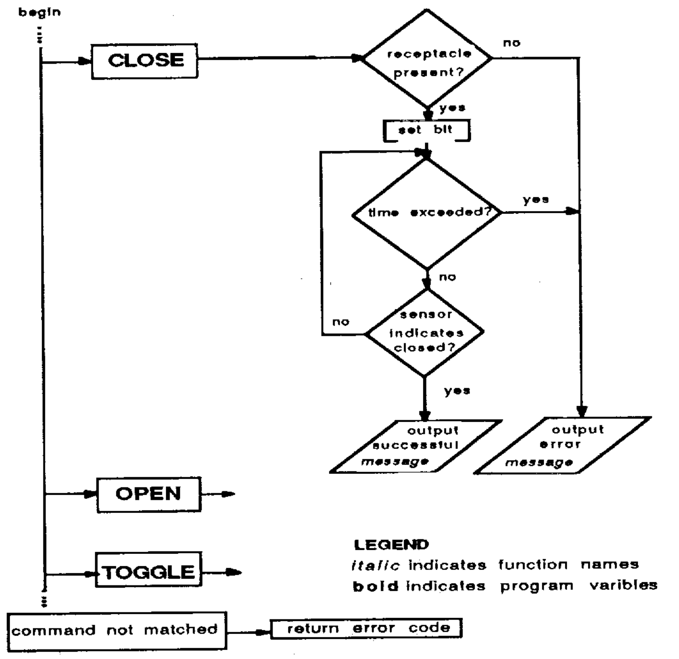

Figure 10
Block Diagram of Function to Execute Typical Component Command
(Receptacle)

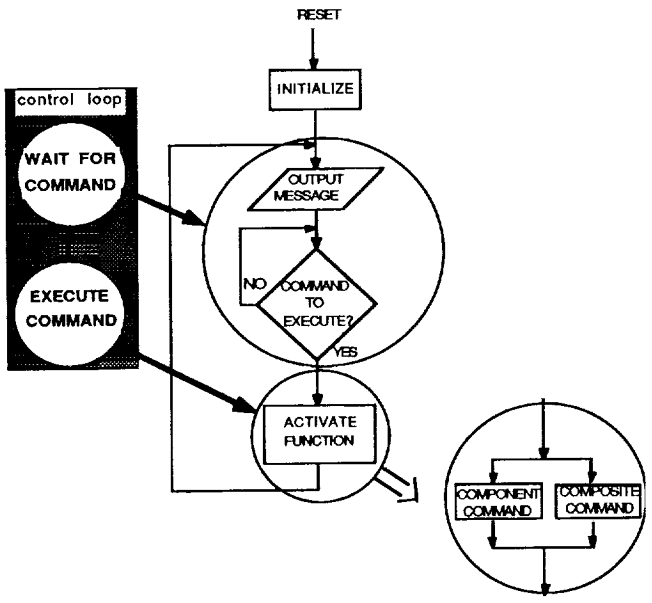

Figure 9
Software Control Loop.

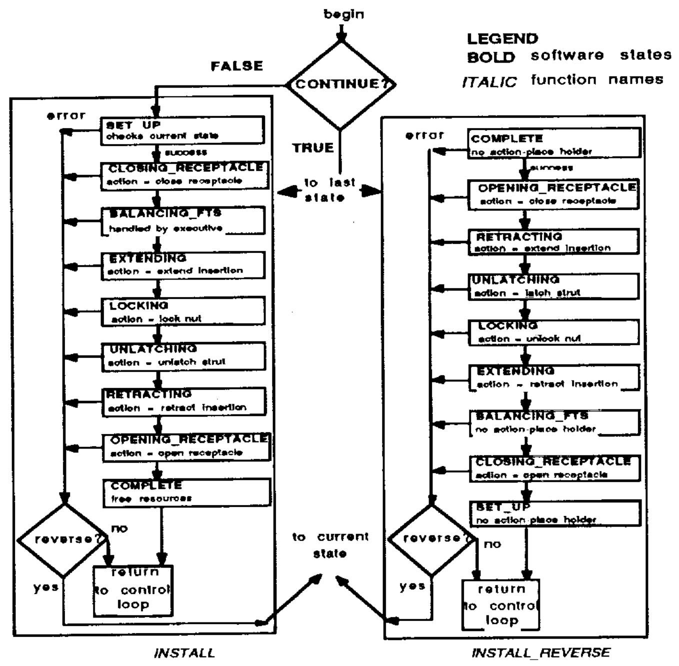

Figure 11
Block Diagram of Function to Execute Typical Composite Command
(Portion of Install)